LISBON SCHOOL OF ECONOMICS & MANAGEMENT
UNIVERSIDADE DE LISBOA

MESTRADO

GESTÃO DE SISTEMAS DE INFORMAÇÃO

TRABALHO FINAL DE MESTRADO

PROJETO

USO DE METODOLOGIAS ÁGEIS NO TESTE DE UMA APLICAÇÃO DE GESTÃO DE PEDIDOS DE APROVISIONAMENTO DE REDE

SANDRA CRISTINA VIEIRA DA CUNHA

NOVEMBRO 2020

LISBON SCHOOL OF ECONOMICS & MANAGEMENT
UNIVERSIDADE DE LISBOA

MESTRADO EM

GESTÃO DE SISTEMAS DE INFORMAÇÃO

TRABALHO FINAL DE MESTRADO

TRABALHO DE PROJETO

USO DE METODOLOGIAS ÁGEIS NO TESTE DE UMA APLICAÇÃO DE GESTÃO DE PEDIDOS DE APROVISIONAMENTO DE REDE

SANDRA CRISTINA VIEIRA DA CUNHA

ORIENTAÇÃO:

JESUALDO CERQUEIRA FERNANDES

NOVEMBRO 2020

LISBON SCHOOL OF ECONOMICS & MANAGEMENT
UNIVERSIDADE DE LISBOA

ÍNDICE

ÍNDICE DE FIGURAS

RESUMO

As metodologias ágeis são cada vez mais utilizadas na indústria de desenvolvimento de software, especialmente pela sua facilidade na incorporação de alterações a requisitos de cliente ao longo de um projeto.

Este tema das metodologias ágeis surge também associado ao mestrado em gestão de sistemas de informação, enquadrando-se o projeto desenvolvido como uma aplicação prática de conceitos estudados.

Adicionalmente, os projetos de desenvolvimento de software têm um ciclo de vida associado e a fase de testes de sistema e de integração foi que mais experienciei durante os projetos que desenvolvi a nível empresarial.

Assim, o objetivo deste projeto passou por evidenciar os testes efetuados ao software, ao passo que é utilizada a metodologia ágil SCRUM em todo o ciclo de vida do projeto empresarial.

Ao longo do trabalho apresentado como prático, são abordadas as suas técnicas no que concerne ao teste de uma aplicação de tickets de aprovisionamento de rede. Os testes são, efetivamente, uma parte importante do projeto, uma vez que têm um grande impacto na qualidade do software entregue ao cliente. Consequentemente, é exposto como são desenhados os casos de teste, enquanto a aplicação está a ser desenvolvida. Para cada requisito, os casos de teste são executados quando os desenvolvimentos estão finalizados.

Os casos de teste apresentados foram desenvolvidos e executados de acordo com os processos apresentados e seguindo os pressupostos da metodologia ágil SCRUM.

O ponto chave de aprendizagem foi efetivamente a comunicação ser um ponto fulcral, quer entre o cliente e a equipa, quer entre os membros da equipa para o projeto ser concluído com sucesso. O feedback do cliente no momento dos testes tem um grande impacto na qualidade do software entregue. Além disso, o facto de fazer os testes de sistema ao longo do projeto, em vez de apenas quando os desenvolvimentos estão terminados para todos os requisitos, permitiu que os tempos de entrega fossem mais curtos.

PALAVRAS-CHAVE

Metodologias Ágeis; SCRUM; Testes de Software; Desenvolvimento de software;

ABSTRACT

Agile methodologies are more commonly used on software development, especially for its easiness on incorporating changes requirements during a project.

The agile methodologies subject is part of the master's degree in management of information systems program, framing the project developed as a practical application of studied concepts.

In addition, software development projects have an associated life cycle, and the system and integration testing phases were what I experienced most during the projects I developed at the business level.

Thus, the objective of this project was to highlight the tests carried out on the software, while the SCRUM agile methodology is used throughout the life cycle of the business project.

Throughout the work presented as practical, its techniques are addressed about testing an application of network provisioning tickets. Tests are effectively an important part of the project, as they have a major impact on the quality of the software delivered to the customer. Consequently, it is exposed how the test cases are designed, while the application is being developed. For each requirement, test cases are run when developments are complete. The test cases presented were developed and executed according to the processes presented and following the assumptions of the agile SCRUM methodology.

The key learning point was effectively communication being a focal point, either between the client and the team, or between team members for the project to be successfully completed. Customer feedback at the time of testing has a major impact on the quality of the software delivered. In addition, the fact that system tests are carried out throughout the project, instead of only when developments are complete for all requirements, allowed delivery times to be shorter.

KEYWORDS

Agile methodologies; SCRUM; Software tests; Software development; Agile testing;

1. INTRODUÇÃO

O ciclo de vida do desenvolvimento do *software* é utilizado na indústria de software, de acordo com Subih et al. (2019), para desenhar, desenvolver, produzir *software* de alta qualidade, de confiança e com uma boa relação de custo-benefício. Esta indústria teve uma grande mudança ao passar do tradicional desenvolvimento para a adoção de metodologias ágeis (Sameen Mirza & Datta, 2019) e existem grandes diferenças nestas duas metodologias (Khan et al., 2017).

O teste completo da aplicação nas metodologias ágeis garante, segundo os autores Khan et al. (2017), que todos os problemas sejam tratados até à entrega seguinte.

Além disso, a pressão está nas pessoas em vez de nos processos/ferramentas e a comunicação com clientes e entre membros da equipa torna-se um ponto chave, fazendo com que também a colaboração do cliente seja mais importante do que a negociação do contrato (Khan et al., 2017).

Durante um projeto que se desenlace com base numa metodologia ágil, é mais importante responder à mudança comparativamente a ter um plano exaustivo para o desenlace do projeto, constatam Khan et al. (2017).

Em segundo lugar, esta indústria está também a ser revolucionada no que diz respeito ao impacto que os testes têm para o sucesso dos produtos de software (Ammann & Offutt, 2017).

Este projeto decorre da implementação de automatizações numa aplicação de pedidos de aprovisionamento de rede que são testadas após o seu desenvolvimento.

2. REVISÃO DA LITERATURA

2.1. Metodologias Ágeis

As metodologias ágeis foram criadas para melhorar o processo de desenvolvimento de software e atenuar os problemas do uso das metodologias tradicionais (Jerom et al., 2020), como a rigidez e a documentação extensa (Ríos & Pedreira-Souto, 2019).

Estas metodologias de desenvolvimento de software estão a tornar-se populares na indústria (Choudhary & Rakesh, 2016) e são preferidos às metodologias tradicionais (Jerom et al., 2020). Isto deve-se

essencialmente ao facto de este método ser flexível no que diz respeito a alterações de requisitos (Jerom et al., 2020; Sharma et al., 2012), porque coloca mais foco na qualidade de desenvolvimento do produto, pela sua simplicidade e aumento de conhecimento na incorporação de alterações (Sameen Mirza and Datta, 2019).

Assim, Sharma et.al (2012) definem o processo ágil como "a habilidade tanto de criar, como responder às alterações de requisitos do *software*". Jerom et al. (2020) completa esta definição, dizendo que o desenvolvimento ágil de software é um conjunto de métodos que se baseiam numa combinação de processos de desenvolvimento incrementais e iterativos, nos quais os requisitos e as soluções para um produto são desenvolvidas através da ligação entre equipas funcionais e organizadas.

Já em 2016 existiam várias metodologias ágeis de desenvolvimento de software e 65% das empresas que adotaram estes métodos indicam que usam SCRUM (Choudhary & Rakesh, 2016).

2.1.1. Caraterísticas das metodologias ágeis

Segundo Ríos & Pedreira Souto (2019), as características das metodologias ágeis são as seguintes:

- Flexibilidade perante alterações de requisitos em qualquer fase do ciclo de vida do projeto, sem afetar o processo de desenvolvimento e planeamento;
- Constante comunicação entre a equipa de desenvolvimento, apesar dos ambientes distribuídos. Incluindo reuniões, comunicação das funcionalidades e alterações feitas ao projeto;
- Inclusão do utilizador final ao longo do ciclo de vida do software. Como resultado, os *developers* vão saber as modificações corretas que devem fazer, na altura certa;
- Documentação necessária e precisa. Sem manter uma documentação extensiva, estas metodologias potenciam documentação específica sobre as funcionalidades da aplicação;
- Implementação de artefactos de UML para a elaboração de documentos e modelos de desenho da aplicação.

2.1.2. Benefícios

Apesar de as metodologias ágeis terem sido bem aceites pela indústria de software (Sameen Mirza & Datta, 2019), estas evidenciam forças, limitações e riscos. Os pontos seguintes representam os benefícios da adoção das metodologias ágeis:

- Iterações (Sameen Mirza & Datta, 2019), cada uma caracterizada por análise, *design*, implementação e teste (Sharma et al., 2012).
- Aviso prévio de risco de desenvolvimento (Sameen Mirza & Datta, 2019; Sharma et al., 2012), e redução do risco de calendário associado à engenharia de software tradicional, em que a integração de todas as componentes desenvolvidas é uma única fase mais tardia no projeto (Santos & Correia, 2015).
- Feedback do cliente regular (Sameen Mirza & Datta, 2019), não só ao longo de todo o projeto, mas após de cada iteração, uma vez que esse "mini projeto" é entregue ao cliente passa seu uso e *feedback* imediato (Sharma et al., 2012). Combinada com pequenos *sprints*, uma maior ênfase no *feedback* do cliente levou a uma melhor agilidade e eficiência das equipas na resposta à alteração de requisitos (Ammann & Offutt, 2017; Santos & Correia, 2015). Assim, é assegurada a satisfação do cliente (Sharma et al., 2012), uma vez que o *software* é desenvolvido com base nos seus requisitos.
- *Releases* rápidas, como consequência de uma integração contínua, com entregas em algumas semanas em vez de alguns meses ou anos (Choudhary & Rakesh, 2016); No final do projeto, verifica-se uma entrega de *software* de alta qualidade, dentro do prazo e do orçamento (Sameen Mirza & Datta, 2019);
- Melhorias nos processos internos, reorganizações ou políticas e escolher os projetos mais adequados e favoráveis (Choudhary & Rakesh, 2016);
- Equipas pequenas (Sameen Mirza & Datta, 2019), e por isso também melhor comunicação e coordenação entre os membros da equipa (Choudhary & Rakesh, 2016);
- Processo mais razoável do que as metodologias tradicionais que se traduziam em processos rígidos de desenvolvimento, e foram moderados no ambiente ágil (Choudhary & Rakesh, 2016);

- Redução do tempo e esforço gasto em documentação (Jerom et al., 2020), limitando-te apenas ao mínimo de documentação necessária, i.e., segundo Sharma et al.(2012), a documentação nas metodologias ágeis é curta e depende da equipa ágil. Praticamente contém uma lista das funcionalidades de produto, duração de cada iteração e a data. Normalmente não há documentação sobre o desenho interno do *software*. O tempo que seria despendido numa documentação completa e extensa é utilizado no desenvolvimento, para entregar o projeto no mínimo de tempo possível.

Para usufruir das vantagens de aplicar as metodologias ágeis no desenvolvimento, segundo Choudahary & Rakesh (2016), há um conjunto de hipóteses que são assumidas como verdade:

- Cooperação e relação cara a cara entre clientes e equipa de desenvolvimento;
- Desenvolver e alterar requisitos do projeto;
- Os *developers* têm boas *skills* e experiências individuais.

Se estas premissas não se aplicarem no projeto de desenvolvimento de software, é melhor procurar outras metodologias que se adequem, por forma a obter melhores resultados (Choudhary & Rakesh, 2016).

2.1.3. Limitações e Riscos

Apesar dos benefícios referidos, existem algumas limitações e riscos associados às metodologias ágeis (Choudhary & Rakesh, 2016; Sameen Mirza & Datta, 2019; Sharma et al., 2012).

Primeiramente, estas metodologias dependem muito do envolvimento do utilizador e, por isso, o sucesso do projeto vai depender da sua cooperação e comunicação (Choudhary & Rakesh, 2016; Sharma et al., 2012). Todo o projeto é baseado no envolvimento do cliente, dizem Sharma et al.(2012), porque é desenvolvido de acordo com os requisitos do cliente. Portanto, se o cliente não tiver claro quais as funcionalidades do produto, o processo de desenvolvimento desviar-se-á do seu propósito. De referir que, segundo Sameen Mirza & Datta (2019), há o risco de existir má comunicação e falta de coordenação entre os membros da equipa e com o cliente, caso não seja compreendido o princípio “Cooperação e relação cara a cara entre clientes e equipa de desenvolvimento “ definido por Choudahary & Rakesh (2016).

A segunda imitação que Sharma et al.(2012) apresentam é a falta de documentação. Apesar de efetivamente poupar tempo para desenvolvimento, é também uma grande desvantagem para o *developer*. O desenho da aplicação pode ser modificado vezes sem conta, dependendo das alterações requeridas pelo cliente, e não há um rastreamento por não ser possível manter em detalhe a documentação sobre o *design* e implementação devido à data limite do projeto. Portanto, dada a pouca informação disponível, é difícil para os *developers* que entram na equipa mais tarde entenderem o método utilizado no desenvolvimento da aplicação.

Por outro lado, as contantes alterações de requisitos podem levar a desperdício de tempo e recursos. Por exemplo, quando o software entregue parcialmente não satisfaz o cliente, será feita uma alteração ao requisito e isso significará tempo adicional despendido em análise e desenho da solução, além dos desenvolvimentos adicionais que podem envolver acréscimo de elementos à equipa (Sharma et al., 2012). Esta desvantagem traz consigo associada o risco de haver um aumento de recursos e aumento do custo total do projeto (Sameen Mirza & Datta, 2019).

Finalmente, este tipo de metodologias é melhor para a gestão do projeto do que para o *developer*. Isto significa que as metodologias ágeis ajudam à gestão de projeto a tomar decisões sobre o desenvolvimento do software, definir metas para os *developers* e uma data limite de entrega. No entanto, é difícil para os *developers* cooperarem com todas as alterações a ambientes, design e código a todo o momento.

2.2. Metodologias ágeis existentes

Diferentes tipos de metodologias ágeis ao longo do tempo foram apresentadas (Jerom et al., 2020) e efectivamente, segundo Sharma et al. (2012), várias metodologias ágeis que podem ser implementadas na concretização de um projeto.

As metodologias ágeis estão focadas em diferentes aspetos do ciclo de vida do desenvolvimento de software: uma focam-se nas práticas, enquanto outras focam-se mais em gerir os projetos de software (Sharma et al., 2012).

As mais populares são: o SCRUM, XP (*extreme programming*), DSDM (*Dynamic Software Development Method*), ASD (*Adaptive Software Development*), FDD (*Feature Driven Development*) e Kanban (Jerom et al., 2020; Sharma et al., 2012).

2.2.1. *Extreme Programming (XP)*

O XP é uma metodologia de sucesso no desenvolvimento de software (Sharma et al., 2012) devido ao seu foco na satisfação do cliente e, por isso, também requer o máximo de interação com o cliente para desenvolver o software. Segundo Jerome et al. (2020), aparenta ser uma boa escolha para uma equipa disciplinada e com conhecimento no que diz respeito a desenvolvimento de software. O mesmo autor afirma inclusive que o XP será maioritariamente utilizado para entregar software de alta qualidade.

Assim, esta metodologia divide o ciclo de desenvolvimento de *software* em vários ciclos curtos de desenvolvimento (Sharma et al., 2012) e, por isso, incorpora alterações aos requisitos dos clientes em qualquer fase do ciclo de desenvolvimento do *software* (Jerom et al., 2020; Sharma et al., 2012).

O método de desenvolvimento de software usando o XP inicia-se com a coleção dos requisitos de utilizador/cliente. Dependendo destes requisitos, todo o processo é então dividido em vários pequenos ciclos de desenvolvimento. Portanto, a fase seguinte é planear as iterações, i.e., decidir o número de ciclos, priorizando os requisitos e estimar o esforço requerido para implementar cada ciclo. Cada iteração é então desenvolvida, seguindo o conceito de programação por pares (*pair programming*) como afirmam Sharma et al. (2012) apud Jerom et al. (2020).

O *pair programming* é um estilo de programação no qual dois programadores trabalham lado a lado num só computador (Jerom et al., 2020), continuamente colaborando no mesmo *design*, algoritmo, código ou teste. Um dos membros do par, chamado o condutor (*driver*), escreve/executa a tarefa e ou outro, chamado navegador, observa atentamente o trabalho do condutor à procura de defeitos (Williams & Kessler, 2003). A utilização desta prática implica que o sucesso do projeto esteja maioritariamente dependente da comunicação entre os membros da equipa e com as equipas que conectam diretamente com o cliente para terem feedback regular (Jerom et al., 2020).

No seu paper, Sharma et al (2012) referem que, durante a fase de desenvolvimento, poderão aparecer novos requisitos de utilizador e o plano de iterações deve ser ajustado de acordo com os mesmos. O passo seguinte é testar a última versão à procura de *bugs*, que serão removidos na iteração seguinte. Após cada teste, a verificação do projeto deve ser efetuada, incluindo um feedback sobre quanto trabalho já foi efetuado até esse momento.

O XP introduziu aos *developers* coisas como a programação por pares, a revisão extensiva do código, refatoração do código e ainda o trabalho em espaço aberto. (Sharma et al., 2012)

2.2.2. SCRUM

O SCRUM é uma *framework*[1] ágil que se foca no desenvolvimento de *software* (Subih et al., 2019), utilizada para organizar equipas e trabalhar de forma mais produtiva (Sharma et al., 2012) e com maior qualidade. Assim, é dada uma maior significância às competências de gestão da equipa envolvida, que incluí tanto gestores de projeto, como *developers*.

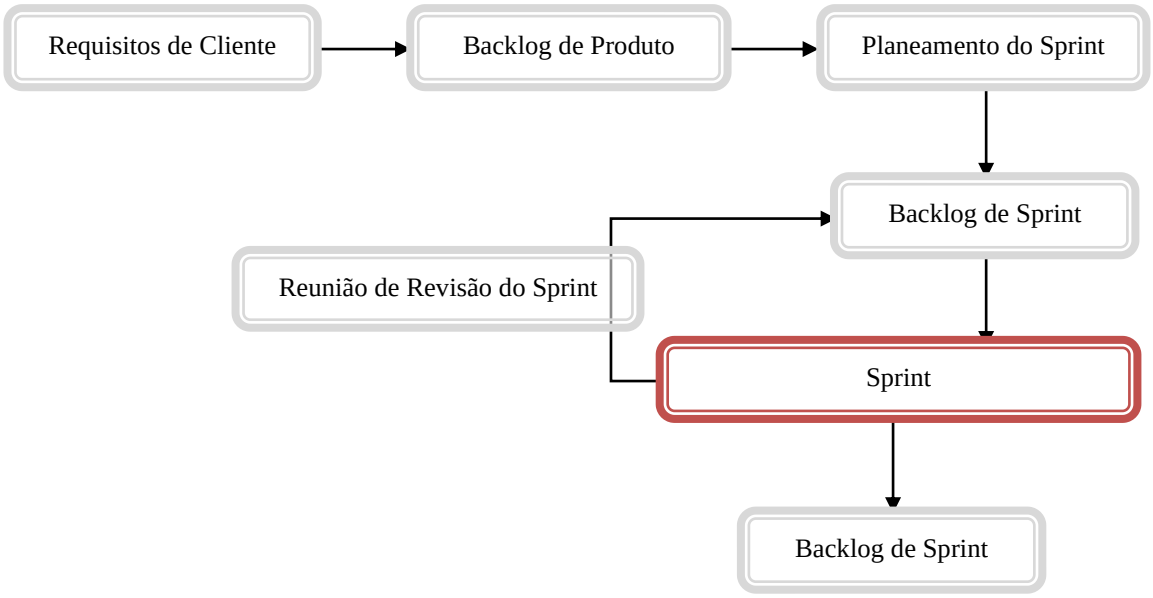

Figura 1. Método de desenvolvimento ágil, utilizando o SCRUM
Fonte: (Sharma et al., 2012)

O processo de desenvolvimento segue o demonstrado na figura 1. Começa por definir os requisitos de utilizador, apesar de não ser esperado que todos os requisitos do projeto sejam ditados pelo utilizador nesta fase (Sharma et al., 2012). O utilizador pode, dada a flexibilidade da metodologia, alterar o seu pedido inicial durante o desenvolvimento, adicionando ou removendo funcionalidades ou atualizando as existentes. Neste momento, existe uma reunião de planeamento (Subih et al., 2019), onde estão presentes

[1] Um sistema de regras, ideias ou crenças que são usadas para planear ou decidir algo (Cambridge Dictionary, n.d.).

os *stakeholders*[2] que são os clientes, *owner* do produto e a equipa SCRUM, por forma a determinar a funcionalidade da aplicação. De seguida, é necessário é priorizar os requisitos e construir o *backlog* de produto (Sharma et al., 2012; Subih et al., 2019).

De acordo com Subih et al. (2019), neste método, os membros da equipa dividem o seu trabalho em ações/tarefas que podem ser completadas dentro de um período predefinido, denominado por *sprints* (Jerom et al., 2020; Sharma et al., 2012). Deve ser feito um planeamento apropriado dos sprints, i.e., quantos sprints são necessários para desenvolver o software, duração do sprint, quais os requisitos do *backlog* de produto que vão ser implementados em cada *sprint* (Sharma et al., 2012). Esta lista de requisitos a serem implementados num sprint denomina-se como *backlog* do *sprint*. Normalmente, cada *sprint* tem a duração de 2 semanas e pode ir até o máximo de 1 mês (Jerom et al., 2020; Sharma et al., 2012; Subih et al., 2019), em que há todos os dias *stand-up meetings* de 15 minutos para dar *feedback* (Sharma et al., 2012; Subih et al., 2019) e planear diariamente o *sprint* (Subih et al., 2019). Estas reuniões diárias (*daily scrums*) são bastante úteis para juntar os *testers* e os *developers* (Choudhary & Rakesh, 2016).

No final de cada sprint, é feita uma revisão, necessária para determinar se todos os requisitos para esse *sprint* específico foram implementados e para decidir quais os que serão implementados no próximo (Sharma et al., 2012).

2.2.3. Feature Driven Development (FDD)

O desenvolvimento direcionado à funcionalidade é um dos métodos de desenvolvimento ágeis (Sharma et al., 2012). Este modelo funciona de forma similar a outros que são incrementais e iterativos, no entanto com variações alguns fatores (Jerom et al., 2020)j.

Segundo os autores Sharma et al. (2012), a maior vantagem deste método é desenhar o domínio do software a ser produzido antes do seu desenvolvimento, uma vez que este modelo se foca maioritariamente nas fases de construção e desenho (Jerom et al., 2020).

[2] Um *stakeholder* é um indivíduo, grupo ou organização que pode afetar, ser afetado ou ter a perceção que é afetado por uma decisão, atividade ou pelo resultado de um projeto (Project Management Institute, 2017).

No FDD começa-se por recolher os requisitos de cliente e construir o modelo geral do projeto, que clarifica o âmbito do *software* (Sharma et al., 2012). De seguida, são listadas as funcionalidades que são valorizadas pelo cliente e agrupadas por domínio. A equipa traça um plano de desenvolvimento das funcionalidades e cada grupo é assignado a uma equipa de desenvolvimento que é liderada por um programador (*chief programmer*). Por fim, as iterações são modeladas, primeiro através de UML e posteriormente desenvolvida a funcionalidade correspondente, repetindo-se este processo até todas as funcionalidades serem implementadas com sucesso.

Ao utilizar este método, dizem os autores Jerom et al. (2020), a entrega dos produtos é rápida e contribuí para uma monitorização perfeita. No entando, os mesmos afirmam que este modelo não é apropriado para projetos muito grandes por funcionar melhor apenas para pequenos projetos.

2.2.4. Dynamic Software Development Method (DSDM)

O método dinâmico de desenvolvimento de *software* é um outro método que se foca na entrega de produto o mais rápido possível, sem afetar a qualidade do produto. Conforme Jerom et al. (2020) afirmam, neste método o ciclo da gestão do projeto pode ser enquadrado com o ciclo de desenvolvimento dos projetos. Isto significa que a maior vantagem do DSDM é criar um protótipo do software o mais cedo possível, podendo ajudar a avançar de imediato para a fase seguinte do projeto, uma vez que desta primeira resulta a análise de risco, priorização das funções da equipa e a documentação sobre o protótipo.

Em conformidade com os mesmos autores, este método é o mais adequado para projetos cujos requisitos de cliente são habitualmente alterados, em vez de projetos com requisitos bem definidos.

Adicionalmente, este método revela um grande controlo sobre a qualidade do produto, fator risco e ainda é eficiente em termos de custos e tempo – o tamanho da equipa varia consoante o tamanho do projeto (Jerom et al., 2020).

Comparativamente a outros modelos, os autores Jerom et al. (2020) consideram este método mais pesado do que outras metodologias ágeis e também mais restritivo.

2.2.5. Adaptative Software Development (ASD)

O desenvolvimento adaptativo de software foca-se maioritariamente em atividades de gestão do projeto. De acordo com Jerom et al. (2020), é baseado numa abordagem incremental e iterativa, portanto os protótipos são elaborados de acordo com o que o utilizador pretende.

O ASD segue 3 fases importantes:

1. Especulação sobre que processo e planeamento serão feitos para o projeto;
2. Colaboração, que foca no desenvolvimento concorrente de produtos ou *features*;
3. Aprendizagem, isto é, rever a qualidade do produto.

Este método é preferível apenas para projetos grandes e complexos (Jerom et al., 2020).

2.3. Desenvolvimento de software utilizando metodologias ágeis

Uma metodologia de desenvolvimento guia o *developer* no processo de desenvolvimento do software (Choudhary & Rakesh, 2016). Segundo os mesmos autores, a metodologia é normalmente escolhida pela equipa de desenvolvimento, dividindo o processo em fases e em cada uma têm de aplicar as diretrizes que metodologia escolhida dita. Selecionar uma metodologia apropriada é importante, consideram Jerom et al. (2020), porque pode maximizar a probabilidade de uma entrega de produto de alta qualidade que cumpre os requisitos do utilizador final.

Entre vários modelos de desenvolvimento de software, Jerom et al. (2020) afirma que as metodologias ágeis são um dos modelos mais populares a serem utilizados na indústria.

No trabalho de desenvolvimento, as equipas focam-se apenas nas funções necessárias de imediato, como declaram Choudhary & Rakesh (2016). A equipa trabalha orientada a uma reposta rápida e flexível, no que diz respeito às alterações de requisitos, entregando-as de forma rápida. Adicionalmente, reúnem feedback e adaptam-se, não só a nível de planeamento, como também à evolução do negócio e tecnologia (Choudhary & Rakesh, 2016; Jerom et al., 2020).

Ríos & Pedreira-Souto (2019) defendem que o desenvolvimento de aplicações baseado numa *framework* de metodologias ágeis está sujeito ao uso dos valores, princípios e práticas ágeis, que favorecem a

construção de aplicações com menos esforço, num menor tempo e com vantagem de ser flexível às alterações ao longo do ciclo de vida.

Assim, de acordo com Herrera e Valência apud. Ríos & Pedreira-Souto (2019), o manifesto ágil aplica 4 valores durante o desenvolvimento de software:

I. Os indivíduos e interações estão acima dos processos e ferramentas;
II. O software em funcionamento antes da documentação: As metodologias ágeis focam-se nos utilizadores envolvidos no sistema e dão prioridade à funcionalidade do software em vez da documentação rigorosa proposta pelas metodologias tradicionais.
III. A colaboração do cliente em vez da negociação do contrato;
IV. Responder à mudança em vez do seguimento de um plano, deixando de lado a inflexibilidade a alterações ao longo do ciclo de vida da aplicação (Ríos & Pedreira-Souto, 2019).

Este manifesto contribuí para a tomada de decisão face aos processos que devem ser seguidos ao longo do projeto.

2.4. Testes e a sua importância

Testar o software é uma fase importante do projeto para garantir o desenvolvimento de software de alta qualidade (Garousi et al., 2020) e o recente crescimento dos processos ágeis coloca pressão acrescida nesta fase (Ammann & Offutt, 2017). Esta é uma das formas de obter informação sobre as qualidades do software (Harty, 2020).

Os testes de software podem ser mensurados de várias formas, incluindo o progresso que foi feito (e.g. o que testámos?) e os resultados obtidos (por exemplo, erros encontrados em testes exploratórios) (Harty, 2020).

O facto de o software ser testado ao longo do processo, em vez de no final, pode trazer vários benefícios para os *developers* da aplicação, especialmente na descoberta de falhas no código ou outros erros. Quando os testes se tornam orgânicos no processo numa base diária, tanto os developers, como a equipa de testes facilmente detetam e corrigem os problemas ao longo do projeto. Esta mecânica garante que o projeto vai continuar e que os próximos problemas vão ser corrigidos mais facilmente. (Santos & Correia, 2015)

Por forma a otimizar este processo, foram desenvolvidos vários procedimentos e ferramentas para gerir a automatização dos testes de sistema e integrar a implementação à medida que vai sendo desenvolvido (Ammann & Offutt, 2017). A automatização de testes não foi aplicada neste projeto, no entanto é referida como sugestão de trabalho futuro no capítulo 6.

Nas metodologias ágeis, são efetuados ao longo do projeto diferentes testes ao software, nomeadamente testes unitários, de integração e de sistema (Sharma et al., 2012).

3. CONTEXTO DO TFM

Este projeto de final de metrado foi desenvolvido com enquadramento empresarial em consultoria informática no ramo de telecomunicações.

O objetivo do projeto empresarial é implementação de automatizações e verificações numa aplicação de pedidos de aprovisionamento de rede.

Segundo a Cisco (2017), aprovisionamento de rede é um processo de configuração para que utilizadores, dispositivos e/ou servidores tenham acesso à rede. A aplicação engloba, assim, o processo desde o pedido do aprovisionamento de rede por parte de uma operadora de telecomunicações cliente (OTC) da entidade à qual esta empresa presta serviços, até ao momento em que o pedido fica finalizado pelo cliente deste projeto, com toda a informação.

De referir que neste projeto de TFM é exclusivamente abordada a componente de testes do projeto empresarial no que diz respeito ao trabalho de aplicação. Isto significa que, apesar de a metodologia apresentada no capítulo 4 explicar a abordagem do projeto empresarial, no capítulo prático de concretização de projeto (capítulo 5) é reportado o processo de testes de sistema e de integração (testes unitários não são efetuados pela equipa de testes, mas pela equipa de desenvolvimento).

4. METODOLOGIA

O projeto empresarial foi desenvolvido através da metodologia ágil SCRUM. No diagrama da figura 2 estão representadas as características do Scrum aplicadas neste projeto.

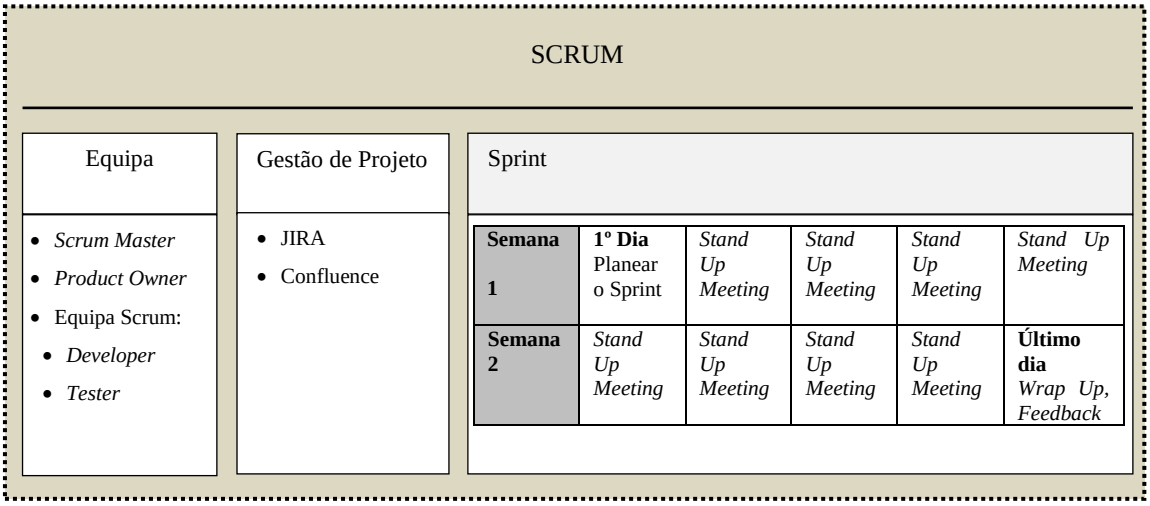

Figura 2. Aplicação da metodologia SCRUM

A equipa é constituída por 1 *scrum master*, 1 *product owner*, 1 *developer* e 1 *tester*.

A plataforma *Confluence* foi utilizada no sentido de organizar os requisitos de cliente em *dashboards*, de forma detalhada, anexando os emails de aprovação do cliente e toda a documentação que foi enviada ao cliente durante o projeto e após a implementação do software.

A gestão do *sprint* foi feita na plataforma JIRA, através da definição da duração do *sprint* (e.g. 2 semanas), tarefas a executar (definição da tarefa, objetivo, sub-tarefas, tempo previsto de execução) e o membro da equipa responsável por cada tarefa/com quem está pendente no momento. Assim, cada membro da equipa sabe as tarefas que tem a cumprir durante o *sprint*. Para isso, existe uma reunião de planeamento, com a duração de 1h30, no primeiro dia de *sprint* e no último uma de verificação das tarefas executadas e objetivos alcançados e ainda dar *feedback* em relação ao trabalho efetuado individualmente e por outros membros da equipa. Nos restantes dias, *stand up meetings* de 10 minutos.

Neste caso, cada *sprint* estava planeado de acordo com a organização prioritária dos requisitos de cliente (descritos na plataforma *Confluence*) já com requisitos de sistema aprovados pelo mesmo, i.e., apesar de existirem 7 requisitos, se o mais prioritário (numa escala de 1 a 5) é o requisito 3, na primeira semana do sprint estão tarefas acerca do desenvolvimento da aplicação seguindo o requisito 3, desenho e execução

de testes de regressão (para perceber o que está implementado e falta implementar) e uma tarefa de desenho dos casos de teste associados aos desenvolvimentos efetuados.

No *sprint* seguinte, as tarefas a executar serão os casos de teste definidos no primeiro, eventualmente são adicionadas tarefas de correção de erros/bugs encontrados nos testes, e neste *sprint* incluí ainda os desenvolvimentos do requisito seguinte na lista de prioridades.

O facto de desenhar, executar testes e relatar erros de implementação por forma a serem corrigidos de imediato, antes dos *User Acceptance Test* (UAT), faz com que haja uma entrega atempada ao cliente nos termos definidos pelo requisito. Quando cada requisito de sistema está totalmente testado e validado, são desenhados os *test-cases* e respetivos *test-steps* para serem efetuados os UAT integrados com o cliente e assim finalizar a aceitação por parte do utilizador, fechando cada requisito de cliente como testado e validado por forma a serem reproduzidos os desenvolvimentos efetuados em ambiente de produção em dia a agendar. Este processo é descrito em detalhe no próximo capítulo.

5. TRABALHO DE APLICAÇÃO

A figura 3 representa o processo da fase de testes até à entrada em produção, que se encontra detalhadamente descrito de seguida. Os pontos 4.1, 4.2 e 4.3 são efetuados enquanto estão a decorrer os desenvolvimentos do requisito de cliente que será testado.

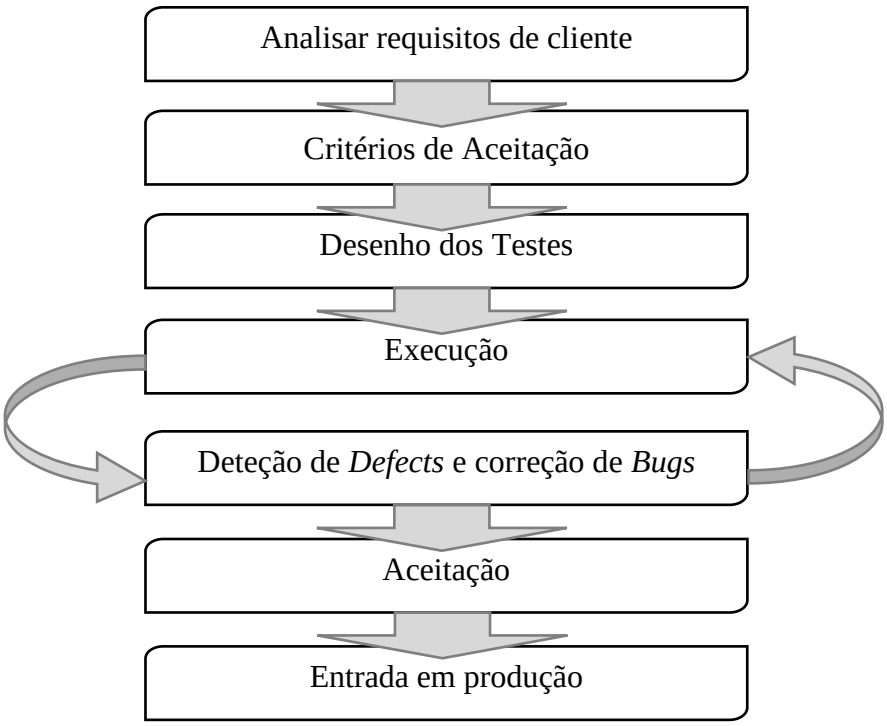

Figura 3. Processo da fase de teste à entrada em produção.

5.1. Identificar requisitos de cliente

Os *user stories requirements* (USR) estão divididos por necessidade, isto é, cada requisito de cliente (Anexo I) traduz-se em várias necessidades que têm associadas requisitos de sistema (Anexos II, III, IV e V). O primeiro passo na fase de testes da aplicação é ler os requisitos de cliente, entender quais as suas expectativas e cruzar essa informação com os requisitos de sistema.

5.2. Definir os critérios de aceitação

Esta fase consiste em delinear os cenários a testar e qual o resultado esperado de cada caso de teste. Nesta fase, é criada uma *dashboard* no *Confluence* na página do requisito de cliente para registar este trabalho e ainda fazer um rascunho de alguns passos do caso de teste. Esta *dashboard* é partilhada com o *developer* que desenvolveu a funcionalidade e é feita análise conjunta dos cenários definidos, não só para complementar já algumas linhas de execução, como para verificar quais os resultados esperados a nível técnico (no *backend*, por exemplo, a população de tabelas na Base de Dados).

5.3. *Desenho dos testes*

Desenhar envolve um detalhe completo dos passos de cada caso de teste. Esta fase de desenho dos testes é escrita numa folha de Excel (representada no Anexo IX) com conexão ao Jira através de uma macro, para efetuar o upload das tarefas. Assim, após os casos de teste estarem finalizados com os respetivos passos de execução, são carregados para o JIRA – desta forma é mais prático do que criar uma tarefa de cada vez no *browser* e preencher os campos um a um.

Os seguintes campos no JIRA são preenchidos: Título do teste (*summary*), descrição, código, dados e/ou informação a utilizar no teste e em cada passo (*input data*), resultados esperados (*expected results*), tempo estimado de execução em cada passo (*estimated time*), componentes (*components* – neste caso são as releases afetadas) , Tipo de Teste (*test cycle group* - testes de sistema ou integrados), prioridade (*priority* - 5 níveis: 0 equivale a prioridade mínima, 5 é prioridade máxima) e qual a *release* que vai ser implementado (*Fix Version*).

Cada caso de teste tem, pelo menos, um link com o requisito de sistema que é testado ao longo desse processo. Escolhe-se um requisito como objetivo principal do teste, e é associado na folha de Excel, para facilitar o *upload*, uma vez que os restantes links têm de ser feitos manualmente (a folha de Excel utilizada apenas permite 1 link por teste). Um requisito de sistema pode ser testado por vários casos de teste e um caso de teste pode testar uma ou vários requisitos de sistema. Associação ao sprint consoante o planeamento do projeto e consoante os desenvolvimentos estão a ser feitos, como mencionado no capítulo anterior.

5.3.1. Caso de teste de exemplo

Apresento de seguida um exemplo prático do trabalho desta fase:

O caso de teste “USR#006.01 - Notificar o Operador nos estados Atribuído e Pendente Operador de Rede”, tem o objetivo de testar a vertente de notificação ao operador móvel quando o pedido está nos estados “Atribuído” e “Pendente Operador de Rede”.

Os resultados esperados deste teste ditam que a ação "Notificar Operador":

- Obriga a introdução de um comentário no campo "Comentário à Ação";
- Não permite a alteração de qualquer outra informação do ticket;

- Ao submeter a ação, o comentário adicionado é enviado ao operador cliente pelo método “Ticket Update”, com a Ação "Envio de Informação Adicional";
- O Ticket regressa ao estado que se encontrava quando foi enviada a Notificação;"

De forma a ficar percetível o que é esperado, as transições entre os estados dos pedidos estão representadas na figura 4.

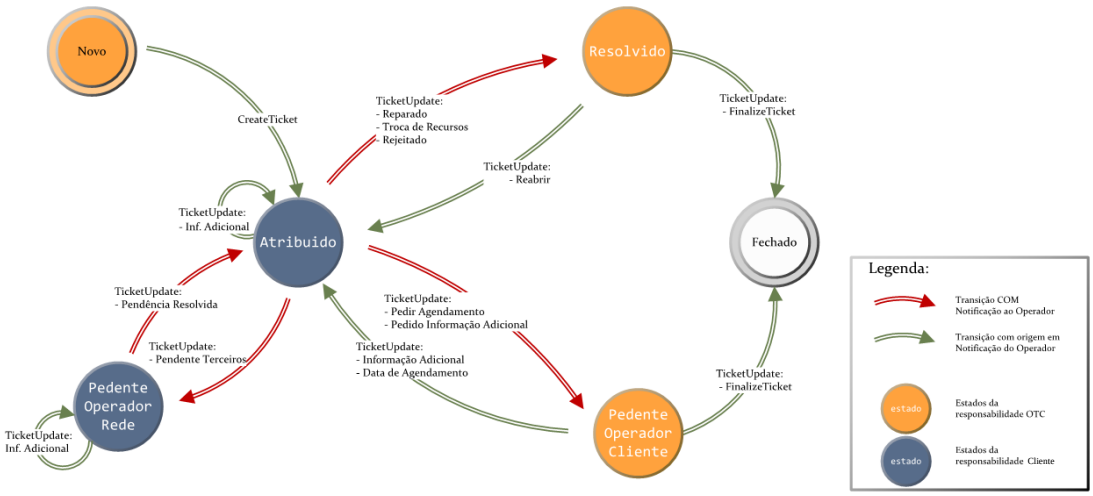

Figura 4. Diagrama de Estados dos pedidos de aprovisionamento.
Fonte: Elaborado pelo SCRUM Master do Projeto.

O total de tempo estimado para execução deste teste é a soma do tempo estimado para cada linha de execução (*test-step*), sendo neste caso de 2 horas e 10 minutos. Este caso de teste tem uma prioridade alta (2 – *High*), está assignado a Sandra Cunha e é executado em detrimento da segunda entrega da *release* 1.6 (*Fix Version: Release 1.6 - Drop2*).

Em cada caso de teste, são desenhados 3 tipos de linhas de execução:

- Verificações a *webservices* utilizados nas transições entre estados;
- Verificações na plataforma *web*, à qual o cliente tem acesso;
- Verificações na Base de Dados, no que diz respeito à população de tabelas associadas ao processo.

5.3.1.1. Desenho da Linha de execução 1 do caso de teste

A primeira linha de execução deste teste é criar um pedido de cliente de um operador móvel através de uma chamada ao *webservice “WS Create”* utilizando o ID de acesso FTTH_DSC_00258594 (que se

encontra na descrição do caso de teste). No caso de existir uma segunda execução do teste, o ID de acesso para ser criado o pedido pode ser obtido através da seguinte consulta que efetuei para retirar o ID anterior na base de dados Oracle:

```
select *
from DSC_t_process_status
where reserve_status ='Y'
  and desactivation_status is null
  and operator ='VDF' AND STATUS ='Terminado'
  and accessid not in (
          select access_id
          from prv_t_order
          where order_reference like 'TT%'
               and status ='R'
               and class ='Incidente Cliente');
```

Esta consulta retorna acessos de rede que se encontram em estado reservado, nunca foram desativados e o seu estado de processo é terminado, ou seja, não está a ser utilizado no momento por nenhum processo que está a correr. Estes são os pressupostos para a criação de cada pedido de aprovisionamento.

Alem deste dado de entrada, defini que é necessário executar a chamada ao *webservice* com os seguintes valores exemplo na composição do *script* em XML:

- Version: 1
- Timestamp: 2019-12-03T15:12:45.948Z
- ProcessID: 2019-12-03
- RequestID: 2019-12-03#XX_XXX
- TrackID: 532192e1-8cd9-4e89-aa98-4bfd145f5a60
- RetryCounter: 0
- SendingOpCo: VDF
- ReceivingOpCo: DSC

- AccessID: FTTH_DSC_XXXXXX (valor indicado na descrição do test case)
- TicketType: Sem Serviços
- TicketSubtype: Pós-Venda
- TicketProblem: Rede Passiva - Sem Sinal
- TicketDescription: Teste Notificar Operador

O resultado esperado desta execução indica que a mensagem do *output* recebido depois da execução do *script* deverá ser de sucesso.

O tempo estimado para esta tarefa é de 5 minutos e será executado pelo utilizador associado, Sandra Cunha.

5.3.1.2. Desenho da Linha de execução 2 do caso de teste

O segundo passo de execução deste caso de teste consiste numa verificação na aplicação *web*. Nesta fase, o pedido encontra-se no estado “Atribuído” (ver Figura 3) e utilizando a ação “Notificar Operador”, o objetivo é testar a vertente do cliente: editar o pedido e atualizá-lo com informação adicional. Para isso, a informação de execução (*input data*) dita para o *tester* abrir o portal *web*, ir à tab "Pendências e Pesquisas", abrir o menu "Pesquisas" e selecionar "Tickets".

1. Pesquisar pelo ID Inicial (ACCESS_ID) ou o Processo Inicial (ORDER_REFERENCE) e executar a pesquisa.
2. Selecionar o ticket e verificar detalhes.
3. Na *datalist*, escolher opção editar ticket. Selecionar a ação "Notificar Operador", adicionar um comentário à ação, por exemplo: "Ação de Notificação em estado Atribuído" e submeter.
4. Após submeter pressionar "Voltar".

Os resultados esperados deste *test-step* são, por ordem de execução:

1. O TT encontra-se disponível para consulta, com estado "Em execução" e o estado do processo "Atribuído" e motivo "Novo".
2. Os detalhes do TT deverão corresponder aos dados utilizados para criar o TT.
3. O ticket deve estar disponível para edição. Apenas os campos Ação e Comentário à Ação devem estar disponíveis para edição, os restantes devem estar bloqueados, não sendo possível fazer

qualquer edição aos mesmos. Deve ser possível submeter a ação escolhida e será apresentada a mensagem "Tarefa Submetida".

4. Após clicar em “Voltar”, deverá ser apresentada a consulta de tickets e o ticket terá o estado "Atribuído" e motivo "Notificar Operador".

O tempo previsto de execução desta verificação é de 15 minutos.

5.3.1.3. Desenho da Linha de execução 3 do caso de teste

A verificação na Base de Dados, no que diz respeito à população de tabelas associadas ao processo é a base do terceiro passo deste caso de teste. O *tester* vai consultar na tabela de pedidos o registo da notificação ao operador decorrente da ação efetuada pelo método “Ticket Update”. O dado de *input* para este passo e a *query* a ser executada:

```
SELECT MESSAGE_XML FROM DSC_T_EXTERNAL_INTERFACE
    WHERE MESSAGE_REF_ID = [order_id do ticket]
        AND message_operation ='Update'
        and sendingopco ='DSC'
        order by modified_date desc;
```

Neste passo, o *tester* deverá substituir a variável “[order_id do ticket]” pela referência do pedido e não deverá exceder os 5 minutos de execução e verificação do resultado esperado na coluna “MESSAGE_XML”:

- o campo <TicketAction> deverá ter o valor "Envio de Informação Adicional";
- o campo <TicketActionDescription> deverá ter o comentário inserido na ação Notificar Operador.

5.3.2. Carregamento dos testes para o JIRA

Depois de todas as linhas de execução de cada teste estarem desenhadas na folha de Excel de carregamento, é efetuado o *upload* do caso de teste e das suas linhas de execução para o JIRA, como demonstra na figura 5.

ID	KEY	Issue Type	Summary	description
	CEE19DSC00300-899	Requirement	Estado Atribuido	
93990	CEE19DSC00300-964	Test Case	USR#006.01 - Notificar o Operador nos estados Atribuido e Pendente Operador de Rede	Notificar o Operador
94007	CEE19DSC00300-966	Test-Step	ar TT Cliente	Criar Ticket de Client
94008	CEE19DSC00300-967	Test-Step	Notificar Operador - Estado Atribuido	Editar ticket e atuali
94009	CEE19DSC00300-968	Test-Step	Consultar tabela - Verificar Ticket Update	Consultar na tabela
94016	CEE19DSC00300-969	Test-Step	Alterar estado TT - Pendente Operador de Rede	Editar ticket e transi
94017	CEE19DSC00300-970	Test-Step	Notificar Operador - Estado Pendente Operador de Rede	Editar ticket e atuali
94018	CEE19DSC00300-971	Test-Step	Consultar tabela - Verificar Ticket Update	Consultar na tabela
94029	CEE19DSC00300-972	Test-Step	Alterar estado TT - TT Atribuido	Editar ticket e transi
94030	CEE19DSC00300-973	Test-Step	Alterar estado TT - TT Pendente Operador Cliente	Editar ticket e transi
94031	CEE19DSC00300-974	Test-Step	Alterar estado TT - TT Atribuido	Transitar ticket para
94032	CEE19DSC00300-975	Test-Step	Alterar estado TT - Resolvido	Editar ticket e transi
94033	CEE19DSC00300-976	Test-Step	Fechar TT - Fechar DST	Fecho do ticket unil
94034	CEE19DSC00300-977	Test-Step	Consultar TT - Verificar Comentários	Verificar Origem dos

Figura 5. Passo 1 do upload: Carregar botão para iniciar

De seguida são inseridas as credenciais de acesso do utilizador e clicar no botão "Upload", como sugere a figura 6.

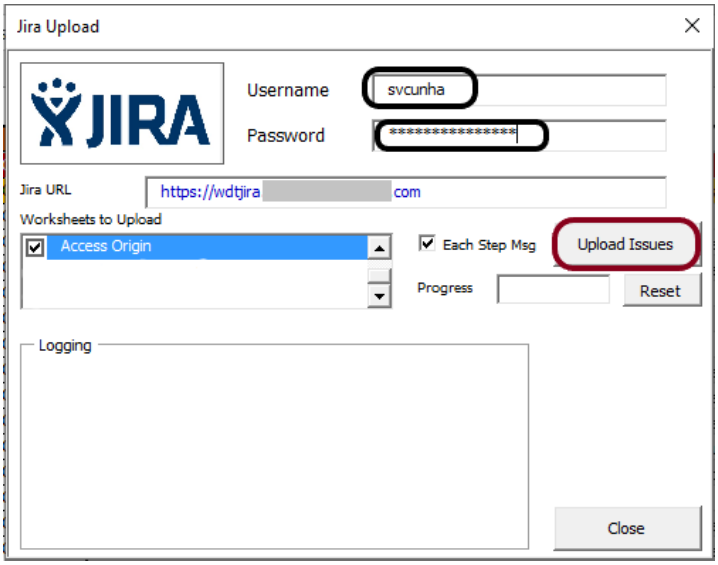

Figura 6. Passo 2 do upload: Inserir credenciais de acesso ao JIRA e iniciar upload.

Os requisitos de sistema "USR#006.01 - Estado Atribuído", "USR#006.01 - Estado Pendente Operador Rede" e "USR#006.01 - Tab comentários: origem do comentário" são testados pelo caso de teste "USR#006.01 - Notificar o Operador nos estados Atribuído e Pendente Operador de Rede", pelo que, quando efetuado o *upload* do teste para o JIRA, é associado um link que menciona que os requisitos são testados por este caso de teste.

Issue Links:	Tests		
	tests CEE19DSC00300-899	USR#006.01 - Estado Atribuido	Tested & Validated
	tests CEE19DSC00300-905	USR#006.01 - Estado Pendente Operador...	Tested & Validated
	tests CEE19DSC00300-912	USR#006.01 - Tab comentários: origem...	Tested & Validated

Figura 7. Requisitos de sistema associados ao caso de teste no JIRA
(esta versão é após exportar o caso de teste do JIRA para uma versão em documento, Anexo VII)

De seguida, o caso de teste é associado ao sprint em que será executado. No caso de exemplo, é o sprint número 19 do projeto [DSCv2 S19.46(18.nov a 29.nov)].

5.4. *Execução dos testes*

Utilizando a informação disponibilizada, a aplicação é testada seguindo os passos de execução (Anexos VII e VIII) dos testes, verificando os resultados esperados. Para isso, foram sempre colocadas evidências do resultado obtido em todos os passos (normalmente printscreens ou output da base de dados, quando aplicável).

5.4.1. Execução do Caso de Teste de Exemplo

Nos pontos seguintes está exposta a execução dos primeiros três *test-steps*, que abrangem os tipos de linhas de execução e verificação mencionadas em 4.3.

5.4.1.1. Execução do Primeiro passo do Caso de Teste

No caso do exemplo apresentado no ponto anterior, a chamada ao *webservice* foi efetuada via SoapUI com os dados apresentados. Depois da execução foi anexada à respetiva tarefa no JIRA a evidência presente na figura 8.

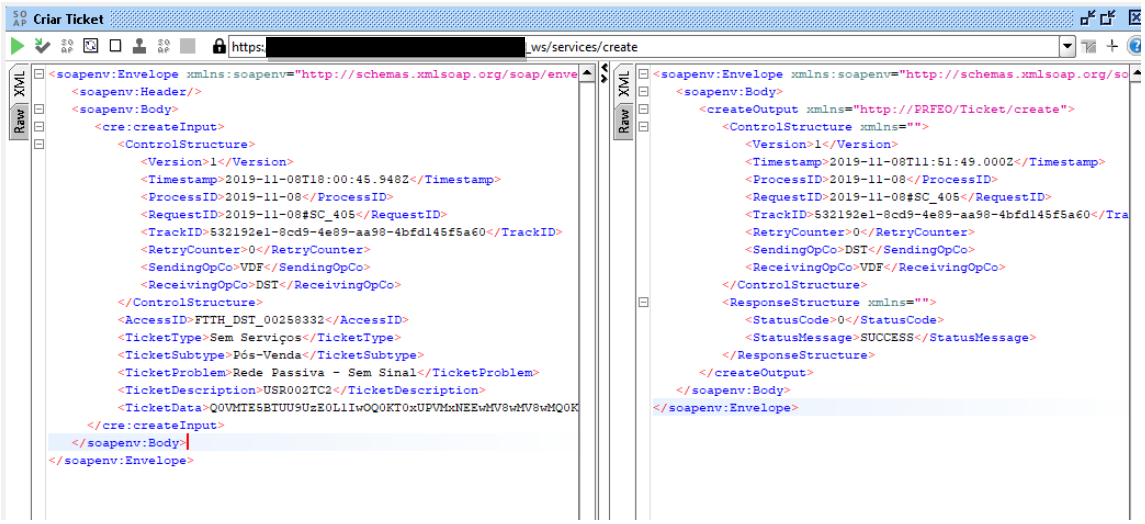

Figura 8. Evidência da primeira linha de execução: chamada ao webservice

Como se pode observar, a resposta do *webservice* retornou sucesso ("StatusMessage = SUCCESS"), significando que o pedido de aprovisionamento foi criado. Portanto, o resultado obtido é igual ao resultado esperado nesta linha de execução. Assim, esta tarefa é marcada do "Testado o Validado" no JIRA.

5.4.1.2. Execução do Segundo passo do Caso de Teste

Seguindo o mesmo exemplo, os resultados do segundo *test-step* são correspondentes ao esperado.

Antes da ação de notificação, no portal *web* o pedido encontra-se disponível para consulta, com estado "Em execução", estado do processo "Atribuído" e motivo "Novo" (figura 9). Os detalhes do pedido correspondem aos dados utilizados para criar o *ticket* (ID de Acesso e Operador) e este encontra-se disponível para edição.

Figura 9. Evidência da segunda linha de execução: consulta do pedido antes da ação de notificação.

Apenas os campos Ação e Comentário à Ação estão disponíveis para edição, os restantes apresentam-se bloqueados, não sendo possível fazer qualquer edição aos mesmos., como sugere a figura 10.

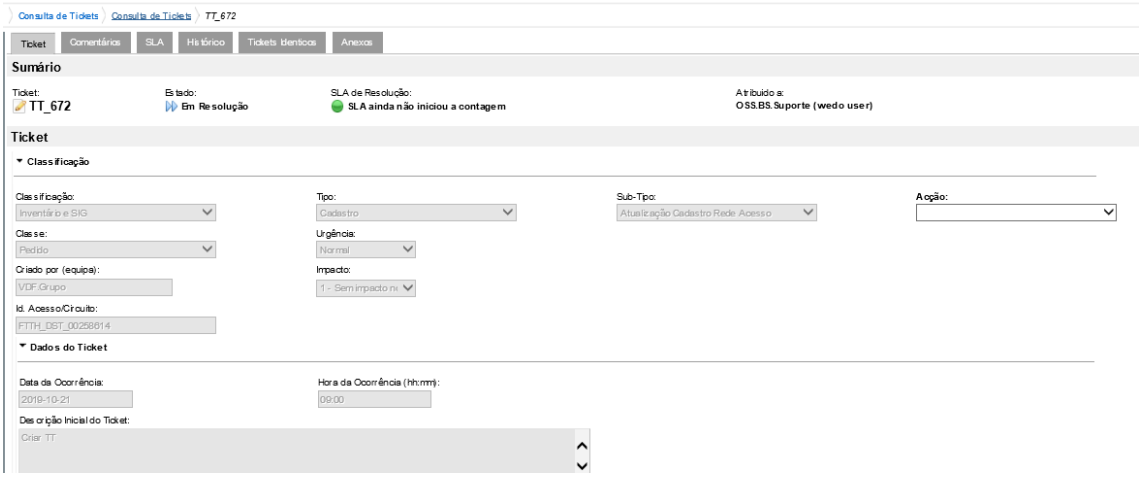

Figura 10. Evidência da segunda linha de execução: Detalhes e Ação a efetuar

Após selecionar a ação “Notificar Operador” e inserir um comentário, foi apresentada a mensagem "Tarefa Submetida" (figura 11).

Figura 11. Evidência da segunda linha de execução: ação de notificação submetida.

Na lista de consulta (figura 12), o pedido encontra-se agora no estado "Atribuído" e motivo "Notificar Operador".

Figura 12. Evidência da segunda linha de execução: Estado do pedido após ação de notificação

Assim, estando cumpridos todos os resultados esperados, este passo do teste é marcado como “Testado e Validado” e foram anexadas todas as evidências necessárias.

5.4.1.3. Execução do Terceiro passo do Caso de Teste

O terceiro passo deste caso de teste é verificar a população de dados numa tabela de controlo em base de dados Oracle (versão 19.2).

Assim, é executada a consulta seguinte:

```
SELECT MESSAGE_XML FROM DSC_T_EXTERNAL_INTERFACE

WHERE MESSAGE_REF_ID = ‘TT_672’

    AND message_operation ='Update'

    AND sendingopco ='DSC'

    order by modified_date desc;
```

O *output* da *query* é uma linha e uma coluna de nome “MESSAGE_XML”, que ao abrir em detalhe foi exportado o seu conteúdo para um ficheiro *.xml* e verifica-se que o resultado obtido é equivalente ao esperado:

```
<TicketStatus>Atribuído</TicketStatus>
<TicketAction>Envio de Informação Adicional</TicketAction>
<TicketActionDescription>Envio de Informação Adicional em estado Atribuído</TicketActionDescription>
<!--Optional:-->
```

Figura 13. Evidência da terceira linha de execução: excerto da extração XML da base de dados

5.5. *Reportar erros*

Sempre e quando é encontrado um erro de implementação ou um *bug*, é reportado à equipa de desenvolvimento, através da criação de um *Defect* no Jira para ser feita a correção. A figura 14 retrata todo o procedimento sobre quando é detetado um erro.

Na criação do *defect*, detalho o erro, preenchendo os seguintes campos: Título/Resumo do erro (deve ser claro e sucinto), descrição detalhada, evidências, data detetada, tipo de problema (se erro de implementação ou bug/limitação da aplicação), componente associada e versão que será implementada a correção. Associo ao sprint, assigno ao *developer* e por vezes estimo o tempo de resolução (baseado na experiência). Defino a severidade do problema (0-Não Bloqueante, 5-Totalmente Bloqueante, i.e., a aplicação está totalmente degradada e as funcionalidades normais não estão acessíveis ao utilizador) e prioridade de resolução (0-Nada prioritário, 5-Prioridade Máxima) consoante seja bloqueante ou não para a execução de um teste (ou mais do que um).

No registo em Jira, quando um *defect* é bloqueante o teste fica sem sucesso e o passo de execução onde foi encontrado o erro fica como testado sem sucesso, sendo associado o *defect* a esse passo. As seguintes linhas de execução ficam no estado de não executadas (*Not Executed*).

Quando o *defect* se encontra no estado *Fixed*, está resolvido no ambiente de desenvolvimento. Depois passa a estado *resolved*, quando já foi feito *deploy* para o ambiente de testes, fica assignado a mim e pronto a retestar. Fecho como *Closed* quando o passo de teste está a retornar o resultado esperado, foi testado e não são necessários mais desenvolvimentos. Um *defect* bloqueante significa que não pode ser continuada a execução do teste por uma componente do requisito de cliente não estar satisfeita ou pela aplicação se encontrar no estado degradado. Neste caso, o teste é reaberto e reexecutado por completo. Caso contrário, é reaberto e o passo de execução que estava sem sucesso é reexecutado e validado, bem como os seguintes.

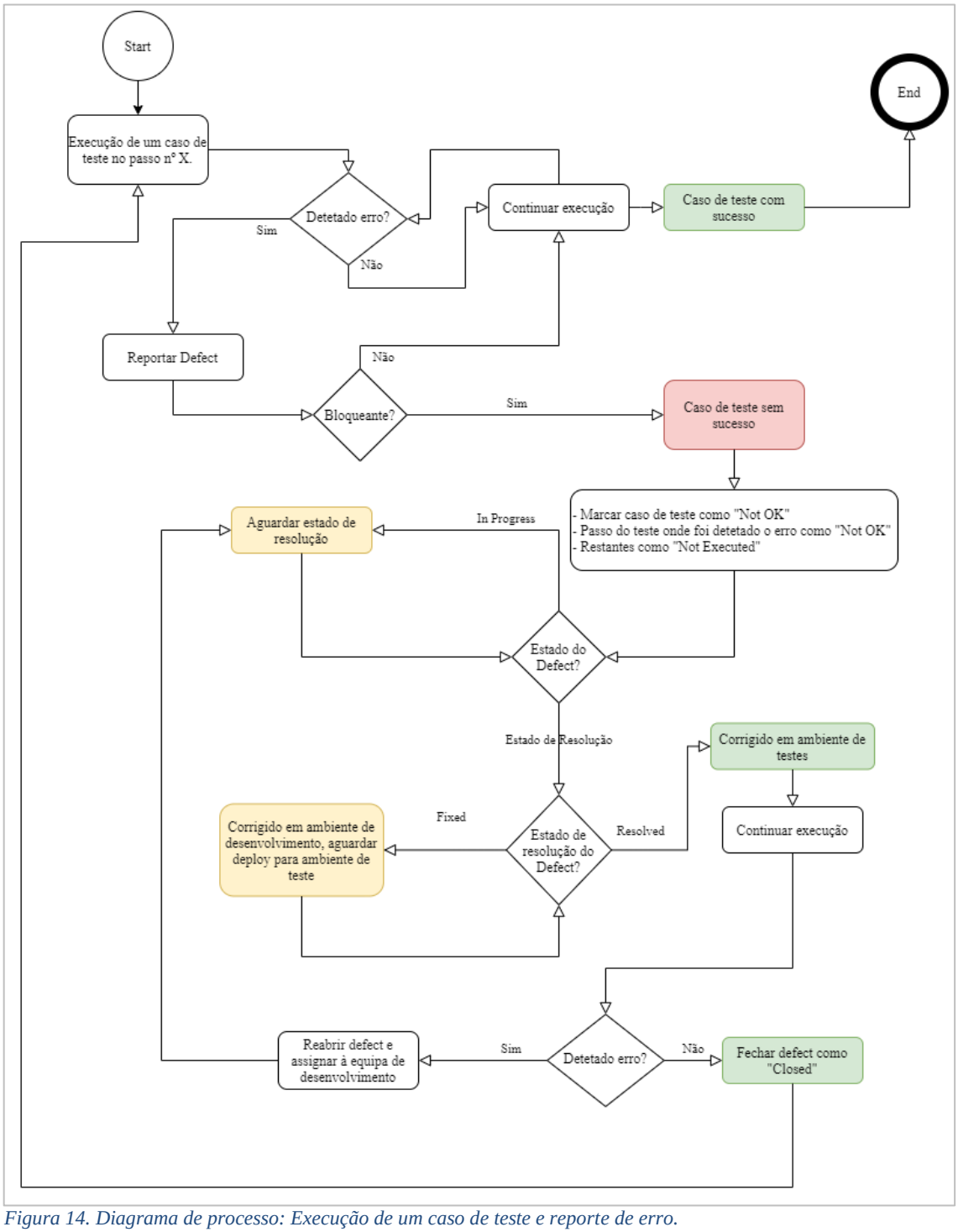

Figura 14. Diagrama de processo: Execução de um caso de teste e reporte de erro.

5.6. *Efetuar testes de aceitação com o cliente*

Os testes de aceitação com o cliente contêm os mesmos cenários dos testes de sistema. No entanto, são descartadas as verificações técnicas de *backend*, como por exemplo, consultas à base de dados, uma vez que está a ser testada a aplicação do ponto de vista do cliente (perspetiva funcional e de negócio). Assim, o desenho dos casos de teste é muito simples, através de uma cópia dos testes de sistema e remoção destes *test-steps* do ponto de vista técnico. Esta fase é feita em folha de *Excel* e efetuado o upload para o JIRA para prosseguir depois à execução.

Neste momento, a aplicação encontra-se num ambiente de pré-produção, ou seja, é uma cópia da aplicação e do sistema *live* que está funcional para testes com o utilizador/cliente antes da entrega final.

Deste modo, os testes são efetuados interagindo com o cliente por chamada via Skype ou através de troca de emails, numa data agendada. Em caso de deteção de um erro de implementação ou um bug, é reportado o erro e após efetuada a correção em ambiente de desenvolvimento, é implementada a nova versão no ambiente de testes de aceitação e de seguida volta-se a testar. Este processo repete-se até à aceitação do cliente estar concluída.

5.7. *Entrega dos desenvolvimentos efetuados*

A entrada do processo em produção é feita no período agendado com o cliente, após a aceitação do cumprimento dos requisitos de forma esperada. No final, são efetuados testes sanitários para concluir a correta instalação e funcionamento da aplicação, que incluem, seguindo o exemplo dado, a verificação da existência e funcionalidade do painel de consulta de pedidos no portal web e ainda a confirmação da criação de tabelas e/ou colunas adicionadas.

6. CONCLUSÃO

As metodologias ágeis são efetivamente cada vez mais uma opção para a indústria de software (Khan et al., 2017; Sameen Mirza & Datta, 2019), pela sua flexibilidade nas alterações de requisitos (Ríos & Pedreira-Souto, 2019), por existir muita comunicação entre todos os envolvidos no projeto e uma maior eficiência na gestão do trabalho (Choudhary & Rakesh, 2016).

No que diz respeito à componente de testes num projeto de desenvolvimento de *software*, esta assume um papel cada vez mais relevante (Garousi et al., 2020) para evidenciar a qualidade do que foi desenvolvido e entregue ao cliente.

Neste projeto, foi implementada a metodologia ágil SCRUM, desde o momento em que se inicia a apresentação dos requisitos pelo cliente até à última entrega. Esta metodologia implicou o trabalho planeado por sprints, de acordo com os requisitos do cliente: em média, os requisitos de sistema associados a um requisito de cliente estavam pronto para a aceitação do cliente num período de 2 sprints. O primeiro para desenvolvimentos e desenho dos cenários e casos de teste e o segundo *sprint* para, efetivamente, testar a aplicação do ponto de vista do sistema. Os testes de aceitação do utilizador/cliente foram efetuados quando o software se encontrava no ambiente de pré-produção e após validação funcional, a entrega foi efetuada na data a agendar. Após a entrega, novos testes foram efetuados, no caso para verificar a disponibilidade dos desenvolvimentos efetuados e a funcionalidade da aplicação.

Em suma, a aplicação desta metodologia permitiu que não só a aplicação fosse testada à medida que os desenvolvimentos estivessem implementados, como também que as entregas fossem ao longo do projeto, diminuído o risco de erros no momento da entrega.

7. LIMITAÇÕES E TRABALHO FUTURO

A utilização de metodologias ágeis acarreta algumas dificuldades, como manter documentados todos os planos de testes e os resultados da execução (Garousi et al., 2020), que senti no desenrolar do projeto na fase de testes, pela quantidade de alterações minuciosas a fazer aos casos de teste e por serem testes exaustivos e serem necessárias evidências em todos os passos da execução. Por outro lado, senti que inicialmente não tinha conhecimento prático suficiente, e ainda hoje tenho muito a evoluir, sendo prova do que menciona Garousi (2020): é necessário mais formação e educação sobre testes de *software*, uma vez que muitos admitem que têm ainda espaço para evoluir.

Neste projeto, a automatização dos testes poderia traduzir-se numa maior eficiência na execução dos mesmos, bem como uma melhor gestão do tempo no projeto, reduzindo o esforço. Segundo Garousi (2020), este é um aspeto-chave para um desenvolvimento de software quando utilizada uma metodologia ágil, por forma a reduzir o tempo entre entregas do *software* ao cliente.

Por outro lado, introduzir desde logo a automatização dos testes (sejam funcionais, carregamento de dados e de performance) como parte dos testes de regressão, significa que a equipa de desenvolvimento assume a responsabilidade de erros no código, diz Santos & Correia (2015), libertando os *testers* para se focarem em casos de teste mais complexos. Os testes unitários são fortemente valorizados e o desenvolvimento orientado a testes faz com que sejam um ponto chave para os requisitos funcionais (Ammann & Offutt, 2017).

REFERÊNCIAS BIBLIOGRÁFICAS

Ammann, P., & Offutt, J. (2017). *Introduction to Software Testing - Paul Ammann, Jeff Offutt - Google Books*. Retrieved August 20, 2020 from https://books.google.pt/books?hl=pt-PT&lr=&id=58LeDQAAQBAJ&oi=fnd&pg=PR10&dq=importance+of+software+testing&ots=Vzl3KIUPZV&sig=zCr51mG43lamofsm78Zu2r2d0UI&redir_esc=y#v=onepage&q&f=true

Cambridge Dictionary. (n.d.). *FRAMEWORK | Significado, definição em Dicionário Inglês*. Retrieved October 30, 2020, from https://dictionary.cambridge.org/pt/dicionario/ingles/framework

Choudhary, B., & Rakesh, S. K. (2016). An approach using agile method for software development. *2016 1st International Conference on Innovation and Challenges in Cyber Security, ICICCS 2016*, 155–158. https://doi.org/10.1109/ICICCS.2016.7542304

Cisco. (2017). *What Is Network Provisioning? - Cisco*. Retrieved November 9, 2020, from https://www.cisco.com/c/en/us/solutions/automation/what-is-network-provisioning.html

Garousi, V., Felderer, M., Kuhrmann, M., Herkiloğlu, K., & Eldh, S. (2020). Exploring the industry's challenges in software testing: An empirical study. *Journal of Software: Evolution and Process*, *32*(8). https://doi.org/10.1002/smr.2251

Harty, J. (2020). How Can Software Testing be Improved by Analytics to Deliver Better Apps? *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, 418–420. https://doi.org/10.1109/ICST46399.2020.00052

Jerom, A., Sp, R., & Scholar, P. G. (2020). *A Survey on Comparative Analysis of Agile Software Development Methodologies*.

Khan, R., Srivastava, A. K., & Pandey, D. (2017). Agile approach for Software Testing process. *Proceedings of the 5th International Conference on System Modeling and Advancement in Research Trends, SMART 2016*, 3–6. https://doi.org/10.1109/SYSMART.2016.7894479

Project Management Institute. (2017). *A guide to the project management body of knowledge (PMBOK guide)* (Sixth). Project Management Institute, Inc.

Ríos, J. M., & Pedreira-Souto, N. (2019). Approach of agile methodologies in the development of web-based software. *Information (Switzerland)*, *10*(10), 21. https://doi.org/10.3390/info10100314

Sameen Mirza, M., & Datta, S. (2019). Strengths and Weakness of Traditional and Agile Processes - A Systematic Review. *Journal of Software*, *14*(5), 209–219. https://doi.org/10.17706/jsw.14.5.209-219

Santos, A., & Correia, I. (2015, May 5). Mobile testing in software industry using agile: Challenges and opportunities. *2015 IEEE 8th International Conference on Software Testing, Verification and Validation, ICST 2015 - Proceedings*. https://doi.org/10.1109/ICST.2015.7102625

Sharma, S., Sarkar, D., & Gupta, D. (2012). *Agile Processes and Methodologies: A Conceptual Study*.

Subih, M. A., Malik, B. H., Mazhar, I., Izaz-ul-Hassan, Sabir, U., Wakeel, T., Ali, W., Yousaf, A., Bilal-bin-Ijaz, Nawaz, H., & Suleman, M. (2019). Comparison of agile method and scrum method with software quality affecting factors. *International Journal of Advanced Computer Science and Applications*, *10*(5), 531–535. https://doi.org/10.14569/ijacsa.2019.0100569

Williams, L., & Kessler, R. R. (2003). *Pair Programming Illuminated*. Retrieved February 8, 2020, from https://books.google.pt/books?hl=pt-PT&lr=&id=LRQhdlrKNE8C&oi=fnd&pg=PR17&dq=pair+programming&ots=UZi7VsROhk&sig=r8xFfnuyvGSbDtWIlg83LAnDMNg&redir_esc=y#v=onepage&q=pair programming&f=false

ANEXOS

I. Requisito de Cliente

[CEE19DSC00300-898] USR#006.01-Tickets de Cliente-Envio de Informação Adicional **Created: 2019-11-18 Updated: 2020-01-06**			
Status:	Tested & Validated		
Project:	CE.E.19.DSC.003.00 - DSC upgrade V2		
Component/s:	Release 1.6		
Fix Version/s:	Release 1.6 - Drop1		
Security Level:	Shared Project Management (Issues shared between Company and the Customer)		
Type:	Epic	Priority:	Medium
Reporter:	SCRUM Master	Assignee:	Sandra
Resolution:	Unresolved	Votes:	0
Labels:	REL1.6		
Attachments:	RE DSC - Release 1#006.01-Informação Adicional em Tickets de Cliente - Para revisão e aprovação.msg RE DSC- Release 1#006.01-Informação Adicional em Tickets de Cliente - Para revisão e aprovação_ok.msg		
Epic Name:	USR#006-Tickets de Cliente		

Description

O requisito atual pretende a implementação de uma *feature* que permita a atualização do ticket, com o envio de informação ao operador cliente por API (quando exista), permanecendo o ticket no mesmo estado em que se encontra. Esta nova ação apenas permitirá adicionar o comentário à ação: nenhuma outra informação será editável.
Esta funcionalidade será aplicada aos estados da responsabilidade do cliente (Atribuído e Pendente de Operador de Rede) e, sendo um operador que use API, será invocado o método update ticket, com a ação "Envio de Informação Adicional".

Generated at Mon Mar 23 10:50:59 GMT 2020 by Sandra using JIRA 7.3.1#73012-sha1:68837e38d8ed1b069612405186dcdceed665bb39.

II. Requisito de Sistema

[CEE19DSC00300-899] USR#006.01 - Estado Atribuído **Created: 2019-11-18 Updated: 2019-12-13**			
Status:	Tested & Validated		
Project:	CE.E.19.DSC.003.00 - DSC upgrade V2		
Component/s:	Release 1.6		
Fix Version/s:	Release 1.6 - Drop2		
Security Level:	Public (Public)		
Type:	Story	Priority:	Medium
Reporter:	SCRUM Master	Assignee:	Developer
Resolution:	Unresolved	Votes:	0
Labels:	REL1.6		
Σ Remaining Estimate:	0 minutes	Remaining Estimate:	Not Specified
Σ Time Spent:	7 hours, 40 minutes	Time Spent:	Not Specified
Σ Original Estimate:	1 day, 4 hours	Original Estimate:	Not Specified

Issue Links:	Tests		
	is tested by	~~CEE19DSC00300-964~~	USR#006.01 - Notificar o Operador nos... Tested & Validated
	is tested by	~~CEE19DSC00300-978~~	USR#006.01 - Comentário em caso de Er... Tested & Validated

Sub-Tasks:	Key	Summary	Status	Assignee
	CEE19DSC00300-902	Configuração de estados e ações	Closed	Sandra
	CEE19DSC00300-903	Alteração de Fluxo do estado atribuído	Closed	Sandra

	CEE19DSC00300-904	Testes Unitários	Closed	Sandra
Epic Link:	USR#006-Tickets de Cliente			
% Done:	100			
Sprint:	DSCv2 S19.46(18.nov to 29.nov), DSCv2 S19.48(02.dez to 13.dez)			
Estimated Effort (ManXDays):	0.8			
Number of Test Cases:	2			

Description

Disponibilização de uma nova ação "Notificar Operador" que:

- Obrigue a introdução de um comentário no campo "Comentário à Ação";
- Não permita a alteração de qualquer outra informação do ticket;
- Ao submeter a ação, o comentário adicionado seja enviado ao operador cliente pelo método Ticket Update, com a Ação "Envio de Informação Adicional";
- O Ticket regressa ao estado atribuído;

Generated at Mon Mar 23 10:48:46 GMT 2020 by Sandra using JIRA 7.3.1#73012-sha1:68837e38d8ed1b069612405186dcdceed665bb39.

III. Requisito de Sistema

[CEE19DSC00300-905] USR#006.01 - Estado Pendente Operador Rede **Created: 2019-11-18 Updated: 2019-12-13**			
Status:	Tested & Validated		
Project:	CE.E.19.DSC.003.00 - DSC upgrade V2		
Component/s:	Release 1.6		
Fix Version/s:	Release 1.6 - Drop2		
Security Level:	Public (Public)		
Type:	Story	Priority:	Medium
Reporter:	SCRUM Master	Assignee:	Developer
Resolution:	Unresolved	Votes:	0
Labels:	REL1.6		
Σ Remaining Estimate:	0 minutes	Remaining Estimate:	Not Specified
Σ Time Spent:	1 hour, 50 minutes	Time Spent:	Not Specified
Σ Original Estimate:	1 day, 4 hours	Original Estimate:	Not Specified

Issue Links:	Tests	
	is tested by	~~CEE19DSC00300-964~~ USR#006.01 - Notificar Operador nos... Tested & Validated

Sub-Tasks:	Key	Summary	Status	Assignee
	CEE19DSC00300-906	Configuração de estados e ações	Closed	Sandra
	CEE19DSC00300-907	Alteração de Fluxo do estado Pendente...	Closed	Sandra
	CEE19DSC00300-908	Testes Unitários	Closed	Sandra

Epic Link:	USR#006-Tickets de Cliente

% Done:	100
Sprint:	DSCv2 S19.46(18.nov to 29.nov), DSCv2 S19.48(02.dez to 13.dez)
Estimated Effort (ManXDays):	0.8
Number of Test Cases:	1
Number of Open Test Cases:	0

Description

Disponibilização de uma nova ação "Notificar Operador" que:

- Obrigue a introdução de um comentário no campo "Comentário à Ação";
- Não permita a alteração de qualquer outra informação do ticket;
- Ao submeter a ação, o comentário adicionado seja enviado ao operador cliente pelo método Ticket Update, com a Ação "Envio de Informação Adicional";
- O Ticket regressa ao estado Pendente de Operador Rede;

Generated at Mon Mar 23 10:50:41 GMT 2020 by Sandra using JIRA 7.3.1#73012-sha1:68837e38d8ed1b069612405186dcdceed665bb39.

IV. Requisito de Sistema

[CEE19DSC00300-912] USR#006.01 - Tab comentários: origem do comentário **Created: 2019-11-18 Updated: 2019-12-13**			
Status:	Tested & Validated		
Project:	CE.E.19.DSC.003.00 - DSC upgrade V2		
Component/s:	Release 1.6		
Fix Version/s:	Release 1.6 - Drop2		
Security Level:	Public (Public)		
Type:	Story	Priority:	Medium
Reporter:	SCRUM Master	Assignee:	Developer

Resolution:	Unresolved	Votes:	0
Labels:	REL1.6		
Σ Remaining Estimate:	0 minutes	Remaining Estimate:	Not Specified
Σ Time Spent:	2 hours, 30 minutes	Time Spent:	Not Specified
Σ Original Estimate:	0 minutes	Original Estimate:	Not Specified

Issue Links:	Tests		
	is tested by ~~CEE19DSC00300-964~~	USR#006.01 - Notificar o Operador nos...	Tested & Validated
	is tested by ~~CEE19DSC00300-978~~	USR#006.01 - Comentário em caso de Er...	Tested & Validated

Sub-Tasks:

Key	Summary	Status	Assignee
CEE19DSC00300-913	Informação adicional ao comentário	Closed	Sandra

Epic Link:	USR#006-Tickets de Cliente
% Done:	100
Sprint:	DSCv2 S19.46(18.nov to 29.nov), DSCv2 S19.48(02.dez to 13.dez)
Number of Test Cases:	2
Number of Open Test Cases:	0

Description

Na tab de comentários do ticket, pretende-se adicionar a identificação clara da origem de cada um dos comentários: "Interno", "Notificação de Operador", "Notificação para Operador"

Generated at Mon Mar 23 10:49:08 GMT 2020 by Sandra using JIRA 7.3.1#73012-sha1:68837e38d8ed1b069612405186dcdceed665bb39.

V. Requisito de Sistema

[CEE19DSC00300-914] USR#006.01 - Tratamento de Erros - Interface com Operador **Created: 2019-11-19 Updated: 2019-12-13**			
Status:	Tested & Validated		
Project:	CE.E.19.DSC.003.00 - DSC upgrade V2		
Component/s:	Release 1.6		
Fix Version/s:	Release 1.6 - Drop2		
Security Level:	Public (Public)		
Type:	Story	Priority:	Medium
Reporter:	SCRUM Master	Assignee:	Sandra
Resolution:	Unresolved	Votes:	0
Labels:	REL1.6		
Σ Remaining Estimate:	0 minutes	Remaining Estimate:	0 minutes
Σ Time Spent:	5 hours, 50 minutes	Time Spent:	Not Specified
Σ Original Estimate:	5 hours	Original Estimate:	1 hour

Issue Links:	Tests		
	is tested by ~~CEE19DSC00300-978~~	USR#006.01 - Comentário em caso de Er...	Tested & Validated

Sub-Tasks:	Key	Summary	Status	Assignee
	CEE19DSC00300-915	Remover a criação de ticket interno	Closed	Sandra
	CEE19DSC00300-916	Erro na Notificação ao Operador	Closed	Sandra

Epic Link:	USR#006-Tickets de Cliente

% Done:	100
Sprint:	DSCv2 S19.46(18.nov to 29.nov), DSCv2 S19.48(02.dez to 13.dez)
Number of Test Cases:	1
Number of Open Test Cases:	0

Description

Tratamento de Erros - Interface com Operador

Pretende-se que a abertura do ticket seja retirada do processo, sendo substituído pela adição de mais um comentário ao ticket referindo o erro ocorrido na interface com o operador.

Generated at Mon Mar 23 10:49:30 GMT 2020 by Sandra using JIRA 7.3.1#73012-sha1:68837e38d8ed1b069612405186dcdceed665bb39.

VI. Requisito - Documentação

[CEE19DSC00300-901] USR#006.01 - Documentação **Created: 2019-11-18 Updated: 2019-12-05**			
Status:	Tested & Validated		
Project:	CE.E.19.DSC.003.00 - DSC upgrade V2		
Component/s:	Release 1.6		
Fix Version/s:	Release 1.6 - Drop2		
Security Level:	Public (Public)		
Type:	Story	Priority:	Medium
Reporter:	SCRUM Master	Assignee:	Sandra
Resolution:	Unresolved	Votes:	0
Labels:	REL1.6		
Σ Remaining Estimate:	0 minutes	Remaining Estimate:	0 minutes
Σ Time Spent:	5 hours, 15 minutes	Time Spent:	30 minutes
Σ Original Estimate:	0 minutes	Original Estimate:	0 minutes

Sub-Tasks:	Key	Summary	Status	Assignee
	CEE19DSC00300-909	Atualização da documentação	Closed	Sandra
	CEE19DSC00300-910	Revisão e Consolidação da Documentação	Closed	SCRUM Master
Epic Link:	USR#006-Tickets de Cliente			
% Done:	100			
Sprint:	DSCv2 S19.46(18.nov to 29.nov)			
Estimated Effort	0.5			

(ManXDays):	

Description

Atualização e revisão da documentação referente a tickets de cliente

Generated at Mon Mar 23 10:54:05 GMT 2020 by Sandra using JIRA 7.3.1#73012-sha1:68837e38d8ed1b069612405186dcdceed665bb39.

VII. Caso de Teste

[CEE19DSC00300-964] USR#006.01 - Notificar o Operador nos estados Atribuído e Pendente Operador de Rede **Created: 2019-11-29 Updated: 2019-12-13 Resolved: 2019-12-13**			
Status:	Tested & Validated		
Project:	CE.E.19.DSC.003.00 - DSC upgrade V2		
Component/s:	Release 1.6		
Fix Version/s:	Release 1.6 - Drop2		
Security Level:	Private (Private)		
Type:	Test Case	Priority:	High
Reporter:	Sandra	Assignee:	Sandra
Resolution:	Test OK	Votes:	0
Labels:	REL1.6		
Σ Remaining Estimate:	0 minutes	Remaining Estimate:	Not Specified
Σ Time Spent:	1 hour, 55 minutes	Time Spent:	Not Specified
Σ Original Estimate:	2 hours, 10 minutes	Original Estimate:	Not Specified

Issue Links:	Tests		
	tests	CEE19DSC00300-899	USR#006.01 - Estado Atribuído Tested & Validated
	tests	CEE19DSC00300-905	USR#006.01 - Estado Pendente Operador... Tested & Validated
	tests	CEE19DSC00300-912	USR#006.01 - Tab comentários: origem… Tested & Validated

Sub-Tasks:	Key	Summary	Status	Assignee

	CEE19DSC00300-966	Criar TT Cliente	Tested & Validated	Sandra
	CEE19DSC00300-967	Notificar Operador - Estado Atribuido	Tested & Validated	Sandra
	CEE19DSC00300-968	Consultar tabela - Verificar Ticket U...	Tested & Validated	Sandra
	CEE19DSC00300-969	Alterar estado TT - Pendente Operador...	Tested & Validated	Sandra
	CEE19DSC00300-970	Notificar Operador - Estado Pendente ...	Tested & Validated	Sandra
	CEE19DSC00300-971	Consultar tabela - Verificar Ticket U...	Tested & Validated	Sandra
	CEE19DSC00300-972	Alterar estado TT - TT Atribuido	Tested & Validated	Sandra
	CEE19DSC00300-973	Alterar estado TT - TT Pendente Opera...	Tested & Validated	Sandra
	CEE19DSC00300-974	Alterar estado TT - TT Atribuido	Tested & Validated	Sandra
	CEE19DSC00300-975	Alterar estado TT - Resolvido	Tested & Validated	Sandra
	CEE19DSC00300-976	Fechar TT - Fechar DSC	Tested & Validated	Sandra
	CEE19DSC00300-977	Consultar TT - Verificar Comentários	Tested & Validated	Sandra
Sprint:	DSCv2 S19.48(02.dez to 13.dez)			

Test Cycle Group:	System Tests
% Test-Step OK:	100
Expected Results:	A ação "Notificar Operador": - Obriga a introdução de um comentário no campo "Comentário à Ação"; - Não permite a alteração de qualquer outra informação do ticket; - Ao submeter a ação, o comentário adicionado é enviado ao operador cliente pelo método Ticket Update, com a Ação "Envio de Informação Adicional"; - O Ticket regressa ao estado que se encontrava quando foi enviada a Notificação;
Number of Defects:	0
Number of Open Defects:	0

Description

Notificar o Operador nos estados Atribuído e Pendente Operador de Rede

AccessID a utilizar na criação do ticket: FTTH_DSC_00258594

Nota: Em caso de retest, obter accessID usando a seguinte consulta:
select * from DSC_t_process_status
where reserve_status ='Y' and desactivation_status is null and operator ='VDF' AND STATUS ='Terminado'
and accessid not in (select access_id from prv_t_order where order_reference like 'TT%' and status ='R' and class ='Incidente Cliente');

Generated at Mon Mar 23 10:47:57 GMT 2020 by Sandra using JIRA 7.3.1#73012-sha1:68837e38d8ed1b069612405186dcdceed665bb39.

VIII. Caso de Teste

[CEE19DSC00300-978] USR#006.01 - Comentário em caso de Erro na Notificação ao Operador **Created: 2019-11-29 Updated: 2019-12-13 Resolved: 2019-12-13**			
Status:	Tested & Validated		
Project:	CE.E.19.DSC.003.00 - DSC upgrade V2		
Component/s:	Release 1.6		
Fix Version/s:	Release 1.6 - Drop2		
Security Level:	Private (Private)		
Type:	Test Case	Priority:	High
Reporter:	Sandra	Assignee:	Sandra
Resolution:	Test OK	Votes:	0
Labels:	REL1.6		
Σ Remaining Estimate:	0 minutes	Remaining Estimate:	Not Specified
Σ Time Spent:	5 hours, 15 minutes	Time Spent:	Not Specified
Σ Original Estimate:	5 hours, 5 minutes	Original Estimate:	Not Specified

Issue Links:	Tests		
	tests	CEE19DSC00300-899	USR#006.01 - Estado Atribuído Tested & Validated
	tests	CEE19DSC00300-912	USR#006.01 - Tab comentários: origem ... Tested & Validated
	tests	CEE19DSC00300-914	USR#006.01 - Tratamento de Erros - In... Tested & Validated

Sub-Tasks:	Key	Summary	Status	Assignee
	CEE19DSC00300-979	Criar TT Cliente	Tested &	Sandra

		Validated	
CEE19DSC00300-980	Notificar Operador - Estado Atribuido...	Tested & Validated	Sandra
CEE19DSC00300-981	Consultar tabela - Verificar Ticket U...	Tested & Validated	Sandra
CEE19DSC00300-982	Comentário - Erro na Notificação ao o...	Tested & Validated	Sandra
CEE19DSC00300-983	Pendencia Terceiros - Pendente Operad...	Tested & Validated	Sandra
CEE19DSC00300-984	Comentário - Erro na Notificação ao o...	Tested & Validated	Sandra
CEE19DSC00300-985	Pendencia Terceiros - Pendente Operad...	Tested & Validated	Sandra
CEE19DSC00300-986	Pendencia Resolvida - Estado Atribuid...	Tested & Validated	Sandra
CEE19DSC00300-987	Comentário - Erro na Notificação ao o...	Tested & Validated	Sandra
CEE19DSC00300-988	Pendencia Resolvida - Estado Atribuido	Tested & Validated	Sandra
CEE19DSC00300-989	Pedir Agendamento - Pendente Op. Clie...	Tested & Validated	Sandra
CEE19DSC00300-990	Verificar que não foi aberto TT interno	Tested & Validated	Sandra
CEE19DSC00300-991	Comentário - Erro na Notificação ao o...	Tested & Validated	Sandra

	CEE19DSC00300-992	Pedir Agendamento - Pendente Op. Cliente	Tested & Validated	Sandra
	CEE19DSC00300-993	Data de Agendamento - Estado Atribuido	Tested & Validated	Sandra
	CEE19DSC00300-994	Verificar TT Atribuido e Comentários	Tested & Validated	Sandra
	CEE19DSC00300-995	Reparar - Estado Resolvido (ERRO)	Tested & Validated	Sandra
	CEE19DSC00300-996	Comentário - Erro na Notificação ao o...	Tested & Validated	Sandra
	CEE19DSC00300-997	Reparar - Estado Resolvido	Tested & Validated	Sandra
	CEE19DSC00300-998	Reabrir - Estado Atribuido	Tested & Validated	Sandra
	CEE19DSC00300-999	Verificar TT Atribuido e Comentários	Tested & Validated	Sandra
	CEE19DSC00300-1000	Reparar - Estado Resolvido	Tested & Validated	Sandra
	CEE19DSC00300-1001	Fechar TT - Fechar DSC	Tested & Validated	Sandra
	CEE19DSC00300-1002	Consultar TT - Verificar Comentários	Tested & Validated	Sandra
Sprint:	DSCv2 S19.48(02.dez to 13.dez)			
Test Cycle Group:	System Tests			
% Test-Step OK:	100			

Number of Defects:	0
Number of Open Defects:	0
Number of Remote Issues:	0

Description

Comentário em caso de Erro na Notificação ao Operador.
AccessID a utilizar na criação do ticket: FTTH_DSC_00258594
Nota: Em caso de retest, obter accessID usando a seguinte consulta:
select * from DSC_t_process_status
where reserve_status ='Y' and desactivation_status is null and operator ='VDF' AND STATUS ='Terminado'
and accessid not in (select access_id from prv_t_order where order_reference like 'TT%' and status ='R' and class ='Incidente Cliente');

Generated at Mon Mar 23 10:49:00 GMT 2020 by Sandra using JIRA 7.3.1#73012-sha1:68837e38d8ed1b069612405186dcdceed665bb39.

IX. Folha de Carregamento de Testes – Caso de Teste de exemplo

ISSUE TYPE	SUMMARY	DESCRIPTION	INPUT DATA	EXPECTED RESULTS	ESTIMATED TIME	PRIORITY
TEST CASE	USR#006.01 - Notificar o Operador nos estados Atribuido e Pendente Operador de Rede	Notificar o Operador nos estados Atribuido e Pendente Operador de Rede AccessID a utilizar na criação do ticket: FTTH_DSC_00258594 Nota: Em caso de retest, obter accessID usando a seguinte consulta: select * from DSC_t_process_status where reserve_status ='Y' and desactivation_status is null and operator ='VDF' AND STATUS ='Terminado' and accessid not in (select access_id from prv_t_order where order_reference like 'TT%' and status ='R' and class ='Incidente Cliente');		A ação "Notificar Operador": - Obriga a introdução de um comentário no campo "Comentário à Ação"; - Não permite a alteração de qualquer outra informação do ticket; - Ao submeter a ação, o comentário adicionado é enviado ao operador cliente pelo metodo Ticket Update, com a Ação "Envio de Informação Adicional"; - O Ticket regressa ao estado que se encontrava quando foi enviada a Notificação;		High
TEST-STEP	Criar TT Cliente	Criar Ticket de Cliente VDF cliente através do WS Create	Executar WS Create com os seguintes valores exemplo: Version: 1 Timestamp: 2019-12-03T15:12:45.948Z ProcessID: 2019-12-03 RequestID: 2019-12-03#XX_XXX TrackID: 532192e1-8cd9-4e89-aa98-4bfd145f5a60 RetryCounter: 0 SendingOpCo: VDF ReceivingOpCo: DSC AccessID: FTTH_DSC_XXXXXX (valor indicado na description do test case) TicketType: Sem Serviços TicketSubtype: Pós-Venda TicketProblem: Rede Passiva - Sem Sinal TicketDescription: Teste Notificar Operador	Mensagem o output recebido depois da execução do WS deverá ser de sucesso	5m	Medium
TEST-STEP	Notificar Operador - Estado Atribuido	Editar ticket e atualizá-lo com informação adicional, utilizando a ação "Notificar Operador".	No portal, na tab "Pendências e Pesquisas", abrir o menu "Pesquisas" e selecionar "Tickets". 1. Pequisar pelo ID Inicial (ACCESS_ID) ou	1. O TT encontra-se disponível para consulta, com estado "Em execução" e o estado do processo "Atribuido" e motivo "Novo".	15m	High

			o Processo Inicial (ORDER_REFERENCE) e executar a pesquisa. 2. Selecionar o ticket e verificar detalhes. 3. Na datalist, escolher opção editar ticket. Selecionar a ação "Notificar Operador", adicionar um Comentário à Ação "Ação de Notificação em estado atribuído" (por exemplo) e submeter. 4. Após submeter pressionar "Voltar".	2. Os detalhes do TT deverão corresponder aos dados utilizados para criar o TT. 3. O ticket deve estar disponível para edição. Apenas os campos Ação e Comentário à Ação devem estar disponíveis para edição, os restantes devem estar bloqueados, não sendo possível fazer qualquer edição aos mesmos. Deve ser possível submeter a ação escolhida e será apresentada a mensagem "Tarefa Submetida". 4. Deverá ser apresentada a consulta de tickets e o ticket terá o estado "Atribuído" e motivo "Notificar Operador".		
TEST-STEP	Consultar tabela - Verificar Ticket Update	Consultar na tabela de pedidos a Notificação do Ticket Update.	Executar a consulta: SELECT MESSAGE_XML FROM DSC_T_EXTERNAL_INTERFACE WHERE MESSAGE_REF_ID = [order_id do ticket] AND message_operation ='Update' and sendingopco ='DSC' order by modified_date desc;	No XML: - o campo <TicketAction> deverá ter o valor "Envio de Informação Adicional" - o campo <TicketActionDescription> deverá ter o comentário inserido na ação Notificar Operador	10m	High
TEST-STEP	Alterar estado TT - Pendente Operador de Rede	Editar ticket e transitar para o estado "Pendente Operador de Rede", utilizando a ação "Pendencia Terceiros".	No portal, na tab "Pendências e Pesquisas", abrir o menu "Pesquisas" e selecionar "Tickets". 1. Pequisar pelo ID Inicial (ACCESS_ID) ou o Processo Inicial (ORDER_REFERENCE) e executar a pesquisa. 2. Selecionar o ticket e escolher opção editar ticket. Selecionar a ação "Pendencia Terceiros" e adicionar um Comentário à Ação "Ação para Pendente Op. de Rede" (por exemplo). 3. Após submeter pressionar "Voltar".	1. O TT encontra-se disponível para consulta, com estado "Em execução" e o estado do processo "Atribuído" e motivo "Notificar Operador". 2. O ticket deve estar disponível para edição. Deve ser possível submeter a ação escolhida e será apresentada a mensagem "Tarefa Submetida". 3. Deverá ser apresentada a consulta de tickets e o ticket terá o estado "Pendente Operador de Rede" e motivo "Pendencia Terceiros".	10m	Medium
TEST-STEP	Notificar Operador - Estado Pendente Operador de Rede	Editar ticket e atualizá-lo com informação adicional, utilizando a ação "Notificar Operador".	1. Na *datalist*, escolher opção editar ticket. Selecionar a ação "Notificar Operador", adicionar um Comentário à Ação "Ação de Notificação em estado atribuído" (por exemplo) e submeter. 2. Após submeter pressionar "Voltar".	1. O ticket deve estar disponível para edição. Apenas os campos Ação e Comentário à Ação devem estar disponíveis para edição, os restantes devem estar bloqueados, não sendo possível fazer qualquer edição aos mesmos. Deve ser possível submeter a ação escolhida e será apresentada a mensagem "Tarefa Submetida". 2. Deverá ser apresentada a consulta de tickets e o ticket terá o estado "Pendente Operador de Rede" e motivo "Notificar	15m	High

				Operador".		
TEST-STEP	Consultar tabela - Verificar Ticket Update	Consultar na tabela de pedidos a Notificação do Ticket Update.	Executar a consulta: SELECT MESSAGE_XML FROM DSC_T_EXTERNAL_INTERFACE WHERE MESSAGE_REF_ID = [order_id do ticket] AND message_operation ='Update' and sendingopco ='DSC' order by modified_date desc;	No XML: - o campo <TicketAction> deverá ter o valor "Envio de Informação Adicional" - o campo <TicketActionDescription> deverá ter o comentário inserido na ação Notificar Operador	10m	High
TEST-STEP	Alterar estado TT - TT Atribuido	Editar ticket e transitar para o estado "Atribuído", utilizando a ação "Pendencia Terceiros".	No portal, na tab "Pendências e Pesquisas", abrir o menu "Pesquisas" e selecionar "Tickets". 1. Pequisar pelo ID Inicial (ACCESS_ID) ou o Processo Inicial (ORDER_REFERENCE) e executar a pesquisa. 2. Selecionar o ticket e escolher opção editar ticket. Selecionar a ação "Pendencia Terceiros" e adicionar um Comentário à Ação "Ação para Pendente Op. de Rede" (por exemplo). 3. Após submeter pressionar "Voltar".	1. O TT encontra-se disponível para consulta, com estado "Em execução" e o estado do processo "Atribuído" e motivo "Notificar Operador". 2. O ticket deve estar disponível para edição. Deve ser possível submeter a ação escolhida e será apresentada a mensagem "Tarefa Submetida". 3. Deverá ser apresentada a consulta de tickets e o ticket terá o estado "Pendente Operador de Rede" e motivo "Pendencia Terceiros".	10m	Medium
TEST-STEP	Alterar estado TT - TT Pendente Operador Cliente	Editar ticket e transitar para o estado "Pendente Operador Cliente", utilizando a ação "Pedir Informação Adicional".	No portal, na tab "Pendências e Pesquisas", abrir o menu "Pesquisas" e selecionar "Tickets". 1. Pequisar pelo ID Inicial (ACCESS_ID) ou o Processo Inicial (ORDER_REFERENCE) e executar a pesquisa. 2. Selecionar o ticket e escolher opção editar ticket. Selecionar a ação "Pedir Informação Adicional" e adicionar um Comentário à Ação "Ação para Pendente Op. Cliente" (por exemplo). 3. Após submeter pressionar "Voltar".	1. O TT encontra-se disponível para consulta, com estado "Em execução" e o estado do processo "Atribuído" e motivo "Pendencia Resolvida". 2. O ticket deve estar disponível para edição. Deve ser possível submeter a ação escolhida e será apresentada a mensagem "Tarefa Submetida". 3. Deverá ser apresentada a consulta de tickets e o ticket terá o estado "Pendente Operador Cliente" e motivo "Pedir Informação".	10m	Medium
TEST-STEP	Alterar estado TT - TT Atribuido	Transitar ticket para o estado atribuído através do WS Update	Executar WS Update com os seguintes valores exemplo: Version: 1 Timestamp: 2019-12-03T15:12:45.948Z ProcessID: 2019-12-03 RequestID: 2019-12-03#XX_XXX (deve ser único para cada pedido do operador) TrackID: 532192e1-8cd9-4e89-aa98-4bfd145f5a60 RetryCounter: 0	Mensagem o output recebido depois da execução do WS deverá ser de sucesso	10m	High

			SendingOpCo: VDF ReceivingOpCo: DSC AccessID: FTTH_DSC_XXXXXX (valor indicado na description do test case) TicketID: TBTS_0000XXXX (substituir X pelo order_id do Ticket - 8 digitos no total) TicketStatus: Atribuído TicketAction: Envio de Informação Adicional TicketActionDescription: Este é um comentário do Operador com Info Adicional (por exemplo)			
TEST-STEP	Alterar estado TT - Resolvido	Editar ticket e transitar para o estado "Resolvido", utilizando a ação "Reparar".	No portal, na tab "Pendências e Pesquisas", abrir o menu "Pesquisas" e selecionar "Tickets". 1. Pesquisar pelo ID Inicial (ACCESS_ID) ou o Processo Inicial (ORDER_REFERENCE) e executar a pesquisa. 2. Selecionar o ticket e escolher opção editar ticket. Selecionar a ação "Reparar", Tipificação de Fecho "Vandalismo" (por exemplo) e adicionar um Comentário à Ação "Ação para Resolvido" (por exemplo). 3. Após submeter pressionar "Voltar".	1. O TT encontra-se disponível para consulta, com estado "Em execução" e o estado do processo "Atribuído" e motivo "Enviar Informação Adicional". 2. O ticket deve estar disponível para edição. Deve ser possível submeter a ação escolhida e será apresentada a mensagem "Tarefa Submetida". 3. Deverá ser apresentada a consulta de tickets e o ticket terá o estado "Resolvido" e motivo "Reparar".	10m	Medium
TEST-STEP	Fechar TT - Fechar DSC	Fecho do ticket unilateralmente pela DSC	No portal escolher a tab Operação, aceder ao menu Operação e escolher Fechar Tickets em Estado Operador Na consulta de tickets deverá aparecer o ticket que se encontra “Resolvido”. Nesses registos, pressionar “Fechar Ticket” e na mensagem de aviso escolher “Sim”. Na mensagem de confirmação pressionar “Fechar”. Ver resultado da *datalist* para o ticket.	Ticket com o estado Resolvido deve estar disponível para fecho na *datalist* Consulta de Tickets. O ticket so passará para Resolvido (Fechado) quando o utilizador escolhe a opção “Sim” na mensagem de aviso e uma mensagem de confirmação deverá ser apresentada. Na mensagem de aviso, a opção “Fechar” deverá direcionar o user para a *datalist* de tickets. O Ticket não deverá estar disponível na *datalist*.	10m	Medium
TEST-STEP	Consultar TT - Verificar Comentários	Verificar Origem dos comentários nas transições de estado do ticket.	No portal, na tab "Pendências e Pesquisas" , abrir o menu "Pesquisas" e selecionar "Tickets". 1. Pesquisar pelo ID Inicial (ACCESS_ID) ou o Processo Inicial (ORDER_REFERENCE) e executar a pesquisa. 2. Selecionar o ticket e abrir a tab	1. O TT encontra-se disponível para consulta, com estado "Terminado" e o estado do processo "Resolvido(Fechado)" e motivo "Fechar DSC". 2. Todos os comentários de ações de transição de estados efetuadas pela DSCelecom deverão ter a label [Notificação para o Operador].	15m	High

		"Comentários"	Todos os comentários de ações de transição de estados efetuadas pelo Operador deverão ter a label [Notificação do Operador].